MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

AFOSR-TR. 83-0856

PROCEEDINGS

THIRTEENTH INTERNATIONAL SYMPOSIUM
ON MULTIPLE-VALUED LOGIC

May 23-25, 1983
Kyoto, Japan

IEEE Computer Society   ISMVL-83   ISMVL-Japan   U.S. A.F.O.S.R.

COMPUTER
SOCIETY
PRESS

COMPONENT PART NOTICE

THIS PAPER IS A COMPONENT PART OF THE FOLLOWING COMPILATION REPORT:

(TITLE): Proceedings of the International Symposium on Multiple-Valued Logic (13th)

Held at Kyoto, Japan on May 23-25, 1983.

(SOURCE): _____

To ORDER THE COMPLETE COMPILATION REPORT USE AD-A136 457   .

THE COMPONENT PART IS PROVIDED HERE TO ALLOW USERS ACCESS TO INDIVIDUALLY
AUTHORED SECTIONS OF PROCEEDINGS, ANNALS, SYMPOSIA, ETC. HOWEVER, THE
COMPONENT SHOULD BE CONSIDERED WITHIN THE CONTEXT OF THE OVERALL COMPILATION
REPORT AND NOT AS A STAND-ALONE TECHNICAL REPORT.

THE FOLLOWING COMPONENT PART NUMBERS COMPRISE THE COMPILATION REPORT:

COMPONENT PART NOTICE (CON'T)

COMPONENT PART NOTICE (CON'T)

AD#:	TITLE:
AD-P002 374	Relations among System Diagnosis Models with Three-Valued Test Outcomes.
AD-P002 375	On System Diagnosis with Multivalued Test Outcomes.
AD-P002 376	A Method of Test Generation for Verification of Wiring Correctness.
AD-P002 377	Automated Design of Combinational Networks under Specific Contraints: A Theorem Proving Approach.
AD-P002 378	Synthesis of Multiple-Valued Logic Functions Based on a Modular Design Approach.
AD-P002 379	Synthesis Algorithm for Minimal Components in T-ULM (Universal Logic Modules) Networks.
AD-P002 380	A Fuzzy Relational Inference Language for Expert Systems.
AD-P002 381	The Synthesis of Ternary Functions under Fixed Polarities and Ternary $I^2 L$ Circuits.

Accession For

NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	

By
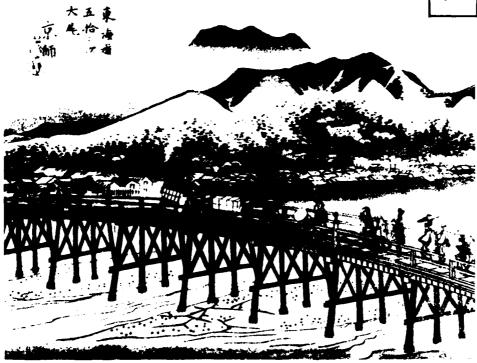
Distribution/

Availability Codes

| Dist | Avail and/or Special |

A-1

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER AFOSR-TR- 83-0836	2. GOVT ACCESSION NO. AD-A136 457	3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle) THE THIRTEENTH INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC	5. TYPE OF REPORT & PERIOD COVERED FINAL REPORT 23 May 83 - 25 May 83
	6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s) J T Butler	8. CONTRACT OR GRANT NUMBER(s) AFOSR-83-0018

9. PERFORMING ORGANIZATION NAME AND ADDRESS Northwestern University Evanston Ill. 60201	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2305/B3

11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NE Building #410 Bolling AFB, Dc 20332	12. REPORT DATE May 1983
	13. NUMBER OF PAGES 431

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION. DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release;
distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Thirteenth International Symposium on Multiple-Valued Logic of 23-25 May 1983 held in Kyoto, Japan was successful.

DD 1 JAN 73 1473   EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

PROCEEDINGS

AFOSR·TR· 83·0836

THE THIRTEENTH INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC

May 23-25, 1983
Holiday Inn
Kyoto, Japan

DTIC
COPY
INSPECTED

SPONSORING ORGANIZATIONS

IEEE Computer Society   ISMVL-83   ISMVL-Japan   U.S. A.F.O.S.R.

Approved for public release;
distribution unlimited.

Additional copies available from:

IEEE Computer Society
P.O. Box 80452
Worldway Postal Center
Los Angeles, CA 90080

IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

COMPUTER SOCIETY PRESS

Published by IEEE Computer Society Press
1109 Spring Street
Suite 300
Silver Spring, MD 20910

Order from: IEEE Computer Society   IEEE Service Center
            Post Office Box 80452   445 Hoes Lane
            Worldway Postal Center   Piscataway, NJ 08854
            Los Angeles, CA 90080

The Institute of Electrical and Electronics Engineers, Inc.

General Chairman's Message

   The 1983 International Symposium on Multiple-Valued Logic
is the thirteenth meeting in a series of annual symposia
devoted exclusively to multiple-valued logic.   Ten of the
past meetings were held in North America, the other two in
Europe.   The ISMVL-83 in Kyoto, Japan, is the first
symposium to be held in Asia.   It is dedicated to the
investigation of multiple-valued logic to narrow the gap
between theory and practice.

   The ISMVL-83 is sponsored jointly by the Multiple-Valued
Logic Technical Committee of the IEEE Computer Society and
ISMVL-Japan.   I would like to express my sincere
appreciation to both of these bodies.

   Planning and organizing for the ISMVL-83 began three years
ago.   It took many meetings and discussions for solving
various problems to reach this stage of welcoming you here
in Kyoto.   We are indebted to all those whose hard work,
dedication,   and enthusiasm have made this symposium
possible.

   I am especially thankful to Professor Y. Tezuka, Chairman
of the Steering Committee, Professor T. Kitahashi, Symposium
Chairman, and Professor J.T. Butler, Symposium Co-Chairman,
for   their   sincere   cooperation,   tireless   work,   and   good
guidance.   Special thanks are also due to Professor M. Goto,
Honorary Member of the General Organizing Committee, for his
continuing   encouragement   and   invaluable   suggestions.   It
should be emphasized that Professor Goto's pioneering work
on multiple-valued logic has led us to hold the ISMVL-83 in
Kyoto, Japan.

   The Program Committee, under the chairmanship of Professor
T. Higuchi with co-chairmen Professor K. Wayne Current and
Dr. S.L. Hurst, has put together the outstanding ISMVL-83
program.   I am grateful to all members of the Program
Committee for their extraordinary effort in organizing the
program.

   This   symposium   could   not   have   been   held   without   the
support of the many Japanese companies that have donated
money to the ISMVL-83. We would like to express our hearty
appreciation to them.

   Finally,   I   would   like   to   thank   the   members   of   the
Organizing Committee of ISMVL-Japan, listed on page vii of
this Proceedings, for their generous support.

Hisashi Mine
Kyoto University

Symposium Chairman's Message

The 1983 International Symposium on Multiple-Valued Logic in Kyoto, Japan, is the first to be held in Asia. Because there are many researchers in Japan and neighboring countries, this symposium offers a unique opportunity for a large number of multiple-valued logic researchers to meet. An Asian conference will encourage the technological development of mainland China, where there are 3000 mathematicians engaged in the study of fuzzy logic. In addition to many speakers from Japan, we have several newcomers from Thailand, China, and Nigeria.

An Asian conference is appropriate for another reason. Sophisticated logics, such as multiple-valued and fuzzy logic, inherit certain characteristics of Oriental philosophy. It is especially appropriate that the first Asian conference be held in Japan, since Professor M. Goto's pioneering work on multiple-valued logic was done here.

This conference is sponsored by the Organization for ISMVL-Japan, the IEEE Computer Society, and the Society's Technical Committee on Multiple-Valued Logic. Operating funds in Japan were provided by many Japanese industries. We are grateful for the generous support of the United States Air Force Office of Scientific Research, which provided funds for the travel of eight U.S. scientists and operating costs in the United States.

We want to recognize the great support of J.T. Butler, Symposium Co-Chairman, and T. Sasao, General Secretary of ISMVL-83, for introducing the Symposium to Japan. Without their efforts, this Symposium could not have been held here.

An essential ingredient of any conference is the technical program. We are truly grateful for the continual help and outstanding work of S.L. Hurst, Program Co-Chairman for Europe. Many papers were submitted by North American researchers, and we are grateful for the efforts of W.C. Current, Program Co-Chairman for North America, for processing these papers. Professor Higuchi, General Program Chairman, deserves a special thank you for handling the many papers from Asian authors, as well as for coordinating the entire program.

Professor Higuchi would like to express his sincere appreciation to the referees, whose intensive perusal and critical comments provided the basis on which decisions of paper acceptance or rejection were based. We also thank the authors, whose research results are the basis for this conference.

Tadahiro Kitahashi
ISMVL-83
Toyohashi University of Technology

v

PREVIOUS PAGE
IS BLANK

Organizing Committee

Chairman: Hisashi Mine
Department of Applied Mathematics and Physics
Kyoto University, Kyoto, Japan

Co-Chairman: Yoshikazu Tezuka
Department of Electrical Communications
Osaka University, Suita, Japan

Members: Motinori Goto   (Meiji University, Japan)
Toshiharu Hasegawa (Kyoto University, Japan)
Tatsuo Higuchi   (Tohoku University, Japan)
Hiroshi Hirayama   (Waseda University, Japan)
Masayuki Kimura   (Tohoku University, Japan)
Tadahiro Kitahashi (Toyohashi University of Technology, Japan)
Yoshiaki Koga   (National Defense Academy, Japan)
Yasuo Komamiya   (Kyushu University, Japan)
Masao Mukaidono   (Meiji University, Japan)
Matsuroh Nakamichi (Chiba University, Japan)
Akira Nakamura   (Hiroshima University, Japan)
Ryosaku Shimada   (Tokushima University, Japan)
Masaichi Tanaka   (Nihon University, Japan)
Tatsuki Watanabe   (Toyo University, Japan)

Executive Committee

Chairman: Yoshikazu Tezuka
Department of Electrical Communications
Osaka University, Suita, Japan

Symposium
Chairman: Tadahiro Kitahashi
School of Information and Computer Sciences
Toyohashi University of Technology, Toyohashi, Japan

Symposium
Co-Chairman: Jon T. Butler
Department of Electrical Engineering and Computer Science
Northwestern University, Evanston, Illinois, U.S.A.

Program
Chairman: Tatsuo Higuchi
Department of Electronic Engineering
Tohoku University, Sendai, Japan

American
Program
Co-Chairman: K. Wayne Current
Department of Electrical Engineering
University of California-Davis, Davis, California, U.S.A.

European
Program
Co-Chairman: Stanley L. Hurst
School of Electrical Engineering
University of Bath, Bath, England

Treasurer: Masao Mukaidono   (Meiji University, Japan)

General
Secretary: Tsutomu Sasao   (Osaka University, Japan)

Publicity: Okihiko Ishizuka   (Miyazaki University, Japan)

Publication: Michitaka Kameyama (Tohoku University, Japan)

PREVIOUS PAGE
IS BLANK

REFEREES

T. Aihara
C.M. Allen
J.R. Armstrong
J.F. Baldwin
A. Beach
R.G. Bennetts
J. Berman
J.E. Beynon
M.A. Breuer
J.T. Butler
C. Carlsson
W. Coy
S.C. Crist
F. Curl
K.W. Current
T.T. Dao
S. Dhar
J.H. Efstathiou
G. Epstein
D. Etiemble
K. Fang
S. Fujita
Y. Fujita
M. Goto
S. Gottwald
S. Guccioni
A.S. Gupta
G. Hachimine
T. Haga
S.B. Haley
T. Hasegawa
T. Hikita
S.J. Hong
J.L. Huertas
S.L. Hurst
S. Imanishi

O. Ishizuka
M. Israel
W.C. Kabat
L.J. Kahout
A. Kalis
M. Kameyama
A. Kandel
M. Karpovsky
M. Katz
L. Kauffman
H.G. Kerkhoff, Jr.
T. Kitahashi
Y. Koga
S.C. Lee
P. Ligomenides
A.W. Maholick
E. Mandani
J. Mangin
E.J. McCluskey
Y. Miki
M. Mizumoto
D.M. Miller
C. Moraga
H.T. Mouftah
M. Mukaidono
S. Muta
N. Muranaka
J. Muzio
M. Nakamichi
A. Nakamura
Y. Ohkura
I. Okumura
S. Ovchinnikov
C.A. Papachristou
P.D. Picton
B.W. Pilsworth

J. Pla
J.H. Pugsley
D. Ralescu
H. Rasiowa
D.C. Rine
A. Rose
I.G. Rosenberg
N. Sanechika
T. Sasao
R. Shimada
C.B. Silio
D. Simovici
B. Sinha
M. Sinutko, Jr.
W.E. Stein
R.E. Swartwout
S. Termini
A. Thayse
M. Togai
N. Tomabechi
E. Trillas
J.G. Tront
M.P. Tull
F. Ueno
Z.G. Vranesic
T. Wesselkamper
L.B. Wheaton
W. Wojcienchowski
A.S. Wojcik
R.R. Yaeger
Y. Yamamoto
K. Yamato
T.C. Yang
L.A. Zadeh
H.J. Zimmermann

TABLE OF CONTENTS

Invited Address

COMPLETENESS FOR UNIFORMLY DELAYED CIRCUITS

I. G. Rosenberg* and T. Hikita**

* C.R.M.A., Université de Montréal, Montréal, P.Q. H3C 3J7, Canada
** Dept. of Math., Tokyo Metropolitan University, Setagaya, Tokyo 158, Japan

Abstract

The paper reports on the progress towards an effective completeness criterion for uniformly delayed multiple-valued combinatorial circuits. In view of previous work by Hikita&Nozaki and Hikita it suffices to study periodic closed spectra. The main tool is the use of polyrelations (= sequences of relations on $\underline{k} := \{0, \ldots, k-1\}$) and certain constructions on polyrelations developed by Hikita. We were able to restrict the search to unary polyrelations (almost solved) and three types of binary polyrelations:

1) period 2^m, ρ_0 bounded order, $\rho_{2^{m-1}}$ its converse and $\rho_i = \iota_2 := \{(a, a) \mid a \in \underline{k}\}$ otherwise,

2) every nonempty component is of the form $\{(a, s(a)) \mid a \in \underline{k}\}$ where s is a permutation of $\underline{k}$; the permutations are interrelated,

3) components are either (i) all equivalences on $\underline{k}$ or (ii) all central or $= \underline{k}^2$. In both cases they have strong properties in terms of intersecting cliques.

1. Introduction

This paper reports on the progress towards an effective completeness criterion for uniformly delayed circuits (this and other rather technical concepts are fully explained in section 2). In the historical retrospective the topic was introduced rather early by Kudrjavcev in 1960 who defined the various basic concepts and gave an effective completeness criterion for the uniformly delayed binary circuits based on precomplete classes [10, 11]. (For the ordinary non-delayed circuits and logic such a criterion was given by Post [25] for $k = 2$, Jablonskii [8] for $k = 3$, the first author for $k > 3$ [26-28] and the idea of a precomplete class by Kuznecov [15, 16].) Some of Kudrjavcev's results were rediscovered by Loomis [17]. Other completeness aspects for delayed circuits were studied by Birjukova and Kudrjavcev [2].

After this early Russian start the focus moved to Japan where Nozaki and his school took up and expanded the study of multiple-valued delayed circuits, to the extent that during the past 12 years all papers in this domain (with the lone exception of [18]) were published by the Japanese school. For uniformly delayed circuits the breakthrough came in Hikita and Nozaki's 1977 paper [7] which reduced the problem to three more manageable types. The first case (type A) is directly solved by Rosenberg's 1965 primality criterion while the third case (type C) was solved by Hikita in 1979 [6]. Meanwhile Hikita also completely classified the ternary case [4] and gave a relational theory for uniformly delayed circuits [5]. This is based on infinite sequences $\rho = (\rho_0, \rho_1, \ldots)$ of relations on the alphabet $\underline{k} := \{0, 1, \ldots, k-1\}$ of the same arity. For such a sequence, called a polyrelation, an n-ary operation f with nonnegative integer delay δ carries $(\rho_i)^n$ into $\rho_{i+\delta}$ for all $i \geq 0$. This concept replaces the preservation of a single relation which is the basic concept in the non-delayed case.

This address reports on the results towards solving the remaining case of periodic spectra (type B) and the corresponding periodic polyrelations. The precomplete classes obtained are rather exceptional as witnessed by the fact that they are determined by at most binary polyrelations. We have succeeded in limiting them to unary periodic polyrelations (almost solved) and binary polyrelations $\rho = (\rho_0, \rho_1, \ldots)$ of period p of the following three types:

1) $p = 2^m$ $(m > 0)$, ρ_0 is a bounded partial order $\leq$, $\rho_{2^{m-1}}$ is $\geq$, and $\rho_i = \iota_2 := \{(a, a) \mid a \in \underline{k}\}$ for $0 < i < 2^m$, $i \neq 2^{m-1}$. Each of these polyrelations gives a precomplete class [11, 18].

2) Every nonempty component is of the form $\{(a, s(a)) \mid a \in \underline{k}\}$ where s is a permutation of $\underline{k}$. The permutations involved are intimately linked and the case is essentially of a group-theoretical nature to be explored in the future.

3) All components $\rho_0, \ldots, \rho_{p-1}$ are either equivalences $\neq \iota_2$ or all are central or $= \underline{k}^2$ (a symmetric relation σ is central if $\iota_2 \subseteq \sigma \subseteq \underline{k}^2$ and $c \times \underline{k} \subseteq \sigma$ for some $c \in \underline{k}$). In both cases we have strong properties in terms of intersecting cliques which are too complex to be explained here.

The full paper seems to be too long to be included in the proceedings and so we opted for a compromise: we list only definitions and propositions. The preprint of the full paper will be available at

2

the Symposium and may be obtained by writing to either of the authors.

The projec· ·or this work was conceived during A. Nozaki's ·. ·. Hikita's short visits to Montreal in 1979 and the bulk of the work carried out during T. Hikita's one month stay in Montreal in August 1980. The partial financial support provided by NSERC Canada operating grant A-9128 and FCAC Québec Subvention d'équipe Eq-0539 is gratefully acknowledged.

Although the uniformity and the completeness concepts are open to discussion as to their practicality and relation to reality the authors feel that this study is justified as the first step in this direction, and perhaps even more by the richness of the mathematical theory involved. Moreover, the first author thinks that the choice of this topic for an invited talk is only appropriate to express his admiration for the very exciting work done by A. Nozaki and his school in this and other areas of multiple-valued logics and their contributions towards the development of many-valued circuits in the host country of the 13th Symposium.

2. Preliminaries

2.1 Switching circuits are built from basic hardware components which we shall henceforth call *gates*. For simplicity each gate (Fig. 1) is a device with a single output and n inputs (n positive integer). The gate receives and emits signals in the same finite *alphabet* which will be identified with $\underline{k} := \{0, \ldots, k-1\}$. If the signal on the i-th input is x_i ($i = 1, \ldots, n$), then the response of the gate is a unique signal completely determined by the n-tuple $(x_1, \ldots, x_n) \in \underline{k}^n$. Denoting this signal by $fx_1 \ldots x_n$ we can describe the functioning of a gate by an n-ary *operation* f on $\underline{k}$ (i.e. a map $\underline{k}^n \to \underline{k}$). For later use $Q^{(n)}$ stands for the set of all n-ary operations on $\underline{k}$ and we put $Q = \bigcup_{n=1}^{\infty} Q^{(n)}$. Thus to each gate carries an operation f describing its behavior.

$$x_0(t) = f(x_1(t-\delta_1), \ldots, x_n(t-\delta_n)) \qquad (1)$$

where $x_i(t)$ ($i = 0, \ldots, n$) are maps from $[0, \infty)$ into $\underline{k}$ and (1) means that the present output depends on the i-th input δ_i time units ago ($i = 1, \ldots, n$). Such gates are called *delayed input devices* (or *d-modules*). For simplicity we assume that all δ_i belong to the set $N = \{0, 1, \ldots\}$ of nonnegative integers. In this paper we go even further and assume that $\delta_1 = \ldots = \delta_n$. Such a gate is called a *uniformly delayed k-device* (or *module*). It is fully described by the pair $(f, \delta) \in \underline{Q} := Q \times N$ called a *k-valued operation* (or *function*) *with delay*.

2.2 Switching circuits are obtained from a collection of gates by attaching outputs of certain gates to inputs of other gates. Again for simplicity we consider only the combinatorial or feedback free switching circuits. The simplest case is the following. We have a gate F described by $(f, \delta) \in \underline{Q}^{(n)} := Q^{(n)} \times N$ and n gates G_i determined by $(g_i, \delta_i) \in \underline{Q}^{(m_i)}$ ($i = 1, \ldots, n$). If we attach the single output of G_i to the i-th input of F ($i = 1, \ldots, n$) the resulting tree-like circuit has $m := m_1 + \ldots + m_n$ external inputs and realizes the operation $h := f \circledast (g_1, \ldots, g_n) \in Q^{(m)}$ defined by

$$h(x_{11}, \ldots, x_{1m_1}, \ldots, x_{n1}, \ldots, x_{nm_n})$$
$$= f(g_1(x_{11}, \ldots, x_{1m_1}), \ldots, g_n(x_{n1}, \ldots, x_{nm_n}))$$

for all $x_{ij} \in \underline{k}$ ($i = 1, \ldots, n$, $j = 1, \ldots, m_i$). The delays are $(\delta+\delta_1, \ldots, \delta+\delta_1, \ldots, \delta+\delta_n, \ldots, \delta+\delta_n)$ and it follows that the circuit will have a uniform delay if and only if $\delta_1 = \ldots = \delta_n$. Our wish being to stay within uniform delays, we only accept the $\circledast$-composition $(f \circledast (g_1, \ldots, g_n), \delta + \delta')$ with $F := (f, \delta) \in \underline{Q}^{(n)}$ and $G_i := (g_i, \delta') \in \underline{Q}^{(m_i)}$ ($i = 1, \ldots, n$). Denote the resulting circuit by $F \circledast (G_1, \ldots, G_n)$.

Suppose we have a circuit in the shape of a rooted tree with gates at the vertices distinct from the leaves and external inputs (nor necessarily

Figure 1.

In reality the physical time dependent signal $x_i(t)$ on the i-th input ($1 \leq i \leq n$) and the output signal $x_0(t)$ are continuous functions of time. The real situation may be rather complex and so we approximate it by assuming that there are time invariant *delays* $\delta_1, \ldots, \delta_n$ such that

Figure 2.

3
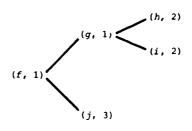
pairwise distinct) at the leaves. Let the sum of the delays be constant on each branch from a leaf to the root. Working from the leaves to the root we can express the delayed function represented by the tree through repeated decomposition (e.g. in the situation of Fig. 2 the function is $(f \circledcirc (g \circledcirc (h, i), j), 4)$). Thus the $\circledcirc$-composition suffices for the description of functions associated to combinatorial circuits yielding uniform delays.

It should be stressed that $\circledcirc$-composition can be performed iff the inside functions have an identical delay. This restriction differentiates our structure from universal algebras and propositional calculus of most logics in which an unrestricted composition is allowed. However the structure may be described as a suitable partial algebra. As this fact seems to have little impact on completeness, we should not dwell on it.

2.3 In what follows we need the *projection* (trivial operation) e_i^n. This is an *n*-ary operation on $\underline{k}$ such that $e_i^n x_1 \ldots x_n = x_i$ for all $x_1, \ldots, x_n \in \underline{k}$.

Let $J := \cup_n \{e_i^n \mid 1 \leq i \leq n\}$ denote the set of all projections. For a subset V of $\underline{U}$ define $<<V>>$ as the least subset of $\underline{U}$ containing $F \circledcirc (G_1, \ldots, G_n)$ whenever $F \in V \cup <<V>>$ and $G_i \in <<V>> \cup (J \times 0)$ $(i = 1, \ldots, n)$. We have added $J \times 0$ to allow arbitrary changes of variables (i.e. for $F \in <<V>>$ the set $<<V>>$ contains also each F' obtained from F by permuting or identifying (fusing) the variables). Clearly $V \mapsto <<V>>$ is a closure operator on $\underline{U}$. The subsets V of $\underline{U}$ satisfying $V = <<V>>$ are called *closed uniform classes*. A closed uniform class containing $J \times 0$ is a *uniform clone*. We say that $V \subseteq \underline{U}$ is *complete* if to every $f \in Q$ there is $\delta \in IN$ such that $(f, \delta) \in <<V>>$. This was introduced for $k = 2$ by Kudrjavcev [10, 11] (as completeness in the second sense) and captures the possibility of constructing each operation with some — possibly very large — delay. The object of this paper is to give a universal completeness criterion. Before embarking into the technical details a comment on the relation between our model and reality. As pointed out in 2.1 the input delayed gate is already a considerable simplification. For fast circuits the delays should not be ignored (to do so is tantamount to neglect such well-known phenomena like races or hazards) and therefore input delayed devices constitute the first step in the right direction. The restriction to uniformly delayed devices is all pervasive through the literature (with the exceptions [2, 12, 18, 22]) but it is not altogether clear whether it is motivated by mere convenience or rather a hard fact about today's commercially available gates. In practice often several functions gave to be represented simultaneously which could be used as an argument for uniform delays. The completeness concept is open to the obvious criticism: What is the purpose of constructing an F with an enormous delay? We defend our model on the following ground: (1) It

is the simplest possible case, and (2) it makes a nice mathematical theory.

2.4 We conclude this with two minor points. Suppose we have constructed (f, δ) and it happens that f is constant (time independent). Of course, there is no observable delay and we can assume that we have all (f, δ') with $\delta' \in IN$. This is rather academical because usually sources of constant signal are so easy to get and cheap that they can be taken for granted. However it is not accounted for in our model.

Finally, we stress that we are interested in sets of gates with the potential to represent any $f \in Q$ (with some delay and assuming an unlimited supply of each type of gates) but ignore completely the optimality: if f can be represented, what is the cheapest way of representing it. This limitation has a good reason because the problem is notoriously hard, depends on the present technology and labor costs and thus, to be meaningful should be closely tailored to a very specific situation which could become obsolete within a very short time.

2.5 We conclude this section with a completeness criterion. First we say that $P \subseteq Q$ is *primal* if every $f \in Q$ is a composition of operations from P (we reserve this term for operations without delays). Put $e := e_1^1$ (i.e. $ex = x$ for all $x \in \underline{k}$). For $V \subseteq \underline{U}$ and $\delta, n \geq 0$ put

$$V^{(n)} := V \cap \underline{U}^{(n)}, \qquad V_\delta := \{f \mid (f, \delta) \in V\},$$

$$V_{(\delta)} := \bigcup_{m=0}^{\infty} V_{m\delta}.$$

We have ([11] Thm. 4 for $k = 2$, [7] and [18] quoted also in [3] Thm. 7.6 p.121 for $k > 2$):

Proposition 2.6. *A closed subset V of $\underline{U}$ is complete if and only if $e \in V_\delta$ and $V_{(\delta)}$ is primal for some $\delta \geq 0$.*

Corollary 2.7. *Let V be a closed uniform class. If V is incomplete, then $F := (J \times 0) \cup V$ is a uniform incomplete clone.*

Needless to say that Proposition 2.6 hardly solves the completeness problem and thus we search for a better criterion. This will be based on sequences of relations introduced and elaborated in the next section.

2.8 A short notational remark. The symbol $\subset$ stands for strict inclusion, while $\subseteq$ means inclusion or equality. Whenever possible an *n*-tuple is written $x_1 \ldots x_n$ instead of the more conventional $(x_1, \ldots, x_n)$. The same applies to arguments of maps, functions and operations, e.g. we write fx or $fx_1 \ldots x_n$ instead of $f(x)$ or $f(x_1, \ldots, x_n)$. Sometimes we do not distinguish notationally an element a and the singleton $\{a\}$ writing e.g. $A \backslash a$ and $a \times A$ for $A \backslash \{a\}$ and $\{a\} \times A$.

4

3. Polyrelations

3.1 A subset of $\underline{k}^h$ is called an *h-ary relation* on $\underline{k}$. An infinite sequence $\rho = (\rho_0, \rho_1, \ldots)$ of *h*-ary relations is called an *h-ary polyrelation*. The set of *h*-ary polyrelations is denoted R_h and $R := \bigcup_{h=1}^{\infty} R_h$. For an *h*-ary relation σ and n positive integer let $\sigma^{[n]}$ denote the set of $h \times n$ matrices whose columns are all in σ. For $f \in Q^{(n)}$ and $X \in \sigma^{[n]}$ let $f[X]$ stand for the row vector $(fX_{1*}, \ldots, fX_{h*})$ where X_{i*} denotes the i-th row of X and let $f[\sigma] := \{f[X] \mid X \in \sigma^{[n]}\}$. We say that f *preserves* σ if $f[\sigma] \subseteq \sigma$ and put $\mathrm{Pol}\,\sigma := \{f \in Q \mid f \text{ preserves } \sigma\}$. In universal algebra terms f preserves σ means σ subalgebra of $\langle \underline{k}; f \rangle^h$. We say that $(f, \delta) \in Q$ preserves an *h*-ary polyrelation $\rho = (\rho_0, \rho_1, \ldots)$ if $f[\rho_i] \subseteq \rho_{i+\delta}$ for all $i \geq 0$. We set

$$\mathrm{Pol}_\delta\rho := \{f \in Q \mid f[\rho_i] \subseteq \rho_{i+\delta}, \; i = 0, 1, \ldots\}$$

for each $\delta \geq 0$, and

$$\mathrm{Pol}\,\rho := \bigcup_{\delta=0}^{\infty} \mathrm{Pol}_\delta\rho \times \delta.$$

Example 3.2. Let $\leq$ be an order (a reflexive, transitive and antisymmetric binary relation). Then $M := \mathrm{Pol}\,\leq$ is the set of $\leq$-*monotonic* operations (i.e. $f \in Q^{(n)}$ such that $fx_1 \ldots x_n \leq fy_1 \ldots y_n$ whenever $x_1 \leq y_1, \ldots, x_n \leq y_n$). Similarly $f \in Q^{(n)}$ is $\leq$-*antimonotonic* if $fx_1 \ldots x_n \geq fy_1 \ldots y_n$ whenever $x_1 \leq y_1, \ldots, x_n \leq y_n$. Let A be the set of $\leq$-anti-symmetric operations and let $\mu := (\leq, \geq, \leq, \geq, \ldots)$. Then $\mathrm{Pol}_{2i}\mu = M$ and $\mathrm{Pol}_{2i+1}\mu = A$ for all $i \geq 0$.

We have [5]:

Lemma 3.3. *Let* $\rho = (\rho_0, \rho_1, \ldots)$ *be a polyrelation. Then* $\mathrm{Pol}_0\rho = \bigcap_{i \geq 0} \mathrm{Pol}\,\rho_i$, *and* $\mathrm{Pol}\,\rho$ *is a uniform clone.*

3.4 For an equivalence ε on $\{1, \ldots, h\}$ put

$$\Delta_\varepsilon := \{a_1 \ldots a_h \in \underline{k}^h \mid \text{if } i\varepsilon j \text{ then } a_i = a_j\}$$

(i.e. Δ_ε consists of all h-tuples over $\underline{k}$ constant on each block of ε). The relations Δ_ε are termed *diagonal*. The diagonal relations and ϕ are called *trivial*. It is well-known [1, 26-29] that $\mathrm{Pol}\,\sigma = Q$ iff σ is trivial. We say that a polyrelation ρ is *proper* if $\mathrm{Pol}\,\rho$ is incomplete. We characterize improper polyrelations.

Proposition 3.4. *A polyrelation* $\rho = (\rho_0, \rho_1, \ldots)$ *is improper if and only if*
 (i) *all* ρ_i *are trivial, or*

(ii) *there are* $p > 0$, $m > 0$ *and trivial relations* $\alpha_0, \ldots, \alpha_{p-1}$ *such that*

$$\rho_i \subseteq \rho_{i+p} \subseteq \ldots \subseteq \rho_{i+mp} = \rho_{i+(m+1)p} = \ldots = \alpha_i$$

for all $i = 0, \ldots, p-1$.

We say that $\rho = (\rho_0, \rho_1, \ldots)$ is *periodic*, if

there is $p > 0$ such that $\rho_{i+p} = \rho_i$ for all $i \geq 0$. For a periodic ρ the least $p > 0$ with this property is the *period* of ρ and denoted p_ρ.

Corollary 3.5. *Let* ρ *be a periodic polyrelation with period* p. *Then* ρ *is proper if and only if at least one* ρ_i *is nontrivial.*

3.6 For a given polyrelation ρ put

$$[\rho] := \{\tau \in R \mid \mathrm{Pol}\,\rho \subseteq \mathrm{Pol}\,\tau\}.$$

It would be useful to have a construction ψ (more precisely a map $\psi : R_n \to R_h$) such that $\psi(\rho) \in [\rho]$ for all $\rho \in R_n$. We give such a map. Let $m > 0$, $n > 0$, $p \geq h > 0$, $A = (a_{ij})$ an $m \times n$ matrix over $\{1, \ldots, p\}$ and $b = (b_1, \ldots, b_m) \in \mathbf{N}^m$. Given an n-ary polyrelation $\rho = (\rho_0, \rho_1, \ldots)$ define an *h*-ary polyrelation $\tau = \psi_{Abh}(\rho)$ by letting τ_ℓ ($\ell \geq 0$) consist of all $u_1 \ldots u_h \in \underline{k}^h$ for which there are $u_{h+1}, \ldots, u_p \in \underline{k}$ such that $u_{a_{i1}} \ldots u_{a_{in}} \in \rho_{\ell+b_i}$ for all $i = 1, \ldots, m$. We illustrate it on a few examples.

Examples 3.7. (1) Let $m = 1$, $n = h = p$ and s permutation of $\{1, \ldots, n\}$. Choosing $A = [s(1) \ldots s(n)]$ and $b = (0)$ we get

$$\tau_\ell = \{u_1 \ldots u_n \mid u_{s(1)} \ldots u_{s(n)} \in \rho_\ell\}.$$

(2) Let $m = n = h = 2$, $p = 3$, $a_{11} = 1$, $a_{22} = 2$, $a_{12} = a_{21} = 3$ and $b_1 = 0$, $b_2 = 1$. Then, for a binary polyrelation ρ we get $\tau = (\rho_0 \circ \rho_1, \; \rho_1 \circ \rho_2, \; \ldots)$ where $\circ$ denotes the standard relational product.

The definition is justified by:

Lemma 3.8. *Let* ρ *and* τ *be as in* 3.6. *Then* $\tau \in [\rho]$ *i.e.* $\mathrm{Pol}\,\rho \subseteq \mathrm{Pol}\,\tau$.

3.9 A uniform incomplete clone is *precomplete* if every uniform clone properly containing it is already complete (i.e. a maximal element of the poset of incomplete uniform clones ordered by $\subseteq$). A clone $M \subset Q$ is *maximal* if $M \subset M' \subset Q$ for no clone M'.

The maximal clones are completely known [26-28]. They are of the form $\mathrm{Pol}\,\sigma$ where the relation σ runs through 6 families. For further use we quote a few of them: (1) proper unary relations (i.e. subsets of $\underline{k}$ distinct from ϕ and $\underline{k}$), (2) binary relations $\{as(a) \mid a \in \underline{k}\}$ where s is a permutation with k/p cycles of prime length p, (3) bounded partial orders (transitive, reflexive and symmetric binary relations with a least and greatest element), (4) equivalences, and (5) binary central relations (i.e. σ reflexive, symmetric, $\neq \underline{k}^2$ and such that $c \times \underline{k} \subseteq \sigma$ for some $c \in \underline{k}$).

A set Ξ of proper polyrelations is termed *generic* if each incomplete uniform clone G extends to $\mathrm{Pol}\,\xi$ for some $\xi \in \Xi$. Our task is to find a small generic set optimally such that each $\mathrm{Pol}\,\xi$ is precomplete. Such a system would provide the best general completeness criterion in the sense that for a given F we have only to test whether $F \subseteq \mathrm{Pol}\,\xi$ for all $\xi \in \Xi$. The essential step is Hikita and Nozaki's generic system Ξ_0 [7].

5

For a relation σ put $\sigma^* := (\sigma, \sigma, \ldots)$. We say that σ^* is of *type* A if Polσ is maximal. Proper periodic polyrelations of arity $\leq k$ are said to be of *type* B. For the last type we need the following special binary relations. An equivalence on $P \subseteq \underline{k}$ distinct from $\{aa \mid a \in P\}$ is called a *proper partial equivalence* on $\underline{k}$. A binary polyrelation $(\rho_0, \iota_2, \iota_2, \ldots)$ is of *type* C' if

1) $\rho_0 = c \times \underline{k}$ for some $c \in \underline{k}$, or

2) ρ_0 is a proper partial equivalence, or

3) $\rho_0 = \{as(a) \mid a \in P\}$ where $P \subseteq \underline{k}$ and either

  i) s is a permutation of P of prime order, or
  ii) s is a permutation of $\underline{k}$, $s|P \neq \mathrm{id}_P$, and

$$s(a) \in P \text{ iff } s(a) = a.$$

We have:

Theorem 3.10. [6, 7] *The set Ξ_0 is generic. The uniform clones of type* A *and* C' *are precomplete.*

4. Minimal Polyrelations

4.1 It remains to study the set $\mathcal{B}$ of polyrelations of type B. Let $\mathcal{P}$ denote the set of all polyrelations of type A or C'. A subset Γ of $\mathcal{B}$ is B-*generic* if each $\beta \in \mathcal{B}$ is dominated by some $\xi \in \Gamma \cup \mathcal{P}$. In view of Theorem 3.10 it suffices to study B-genericity. Let Ξ_1 consist of all $\beta \in \mathcal{B}$ such that

$[\beta] \cap \mathcal{P} = \phi$ (where $[\beta]$ is the set of all polyrelations τ such that Pol$\delta\beta \subseteq$ Pol$\delta\tau$, as defined before). Given $\rho = (\rho_0, \rho_1, \ldots) \in \bar{\mathcal{B}}$ let h_ρ and p_ρ (or h and p) stand for its arity and period. Let $h > 2$. An h-ary relation σ is *totally reflexive* if

$\iota_h \subseteq \sigma \subseteq \underline{k}^h$ (where $\iota_h := \{a_1 \ldots a_h \in \underline{k}^h \mid a_i = a_j$ for some $1 \leq i < j \leq h\}$). A polyrelation $\lambda := (\lambda_0, \lambda_1, \ldots)$ is *totally reflexive* if at least one λ_i is totally reflexive. The next theorem is basic.

Theorem 4.1. *If $\rho \in \Xi_1$ then $[\rho]$ contains no totally reflexive polyrelation.*

A nontrivial relation σ is *primitive* if it is the union of diagonal relations, and a polyrelation λ is *primitive* if all λ_i are trivial or primitive, i.e. $\lambda_i = \Delta_{E_i} := \bigcup_{E_i} \Delta_\varepsilon$ where $E_i \subseteq \mathcal{E}_h$ ($i \geq 0$), and $\mathcal{E}_h$ is the set of equivalences on $\{1, \ldots, h\}$.

Lemma 4.2. *If $\rho \in \Xi_1$, then $[\rho]$ contains no primitive periodic polyrelation.*

We need the following lemmas. Let C denote the set of constant operations on $\underline{k}$.

Lemma 4.3. *Let $\rho \in \Xi_1$ satisfy (i) $h_\rho > 2$, (ii) $[\rho]$ contains no proper binary polyrelation and (iii) $C \subseteq \mathrm{Pol}_0\rho$. Then there is $0 < i < p_\rho$ such that for every $\theta \in \mathcal{E}_h$ with exactly two blocks,*

(*) *if $\Delta_\theta \subseteq \rho_0$ then $\Delta_\theta \subseteq \rho_i$.*

For integers x and y let $x \dotplus y$ denote the inte-

ger z such that $0 \leq z < p$ and $z \equiv x + y$ mod p.

Lemma 4.4. *Let ρ satisfy the assumptions of Lemma 4.3 and let $E \subseteq \mathcal{E}_h$ be such that (i) each $\theta \in E$ has exactly 2 blocks and (ii) $\cap E$ is the least equivalence. Then $\Delta_E \not\subseteq \rho_i$ for all $i \geq 0$.*

4.5 Our goal is to find the smallest possible B-generic system. The strategy is the following: given a B-generic Ξ we find $\Xi' \subseteq \Xi$ such that each $\rho \in \Xi \backslash \Xi'$ is dominated by some $\rho' \in \Xi'$. This reduction will be done in several steps. In the first step we reduce the arities. For the ease of presentation $\rho \in \Xi_1$ is *minimal* if all proper $\tau \in [\rho]$ have $h_\tau \geq h_\rho$. Let Ξ_2 be the set of minimal polyrelations. It is almost immediate that Ξ_2 is generic. First we show that Ξ_2 consists of unary and binary polyrelations. In the remainder of the section the polyrelation $\rho = (\rho_0, \rho_1, \ldots)$ denotes a fixed minimal polyrelation of arity h and period p. Recall that

$$\sigma_h := \{a_1 \ldots a_h \in \underline{k}^h \mid a_i \neq a_j \text{ for } 1 \leq i < j \leq h\},$$

$$\iota_h := \underline{k}^h \backslash \sigma_h,$$

and $\omega_h := \Delta_{1 \ldots h} = \{a \ldots a \in \underline{k}^h \mid a \in \underline{k}\}$. We use throughout the notation $\rho_i = \mu_i \cup \nu_i$ where $\mu_i = \rho_i \cap \sigma_h$ and $\nu_i = \rho_i \cap \iota_h$ ($i = 0, \ldots, p-1$). We start out with the following technical lemmas. For an h-ary relation σ and $1 \leq i_1 < \ldots < i_\ell \leq h$ set

$$\mathrm{pr}_{i_1, \ldots, i_\ell} \sigma = \{(a_1, \ldots, a_\ell) \mid$$
$$a_1 = b_{i_1}, \ldots, a_\ell = b_{i_\ell} \text{ for some } (b_1, \ldots, b_h) \in \sigma\}.$$

Lemma 4.6. *If $\theta \in \mathcal{E}_h \backslash \omega_h$, then every $\nu_i \cap \Delta_\theta$ is trivial.*

An h-ary relation σ is *reflexive* if $\sigma \cap \iota_h$ is diagonal or primitive.

Lemma 4.7. *If $h > 2$ then every ρ_i is empty or reflexive.*

Lemma 4.8. *If $\mu_i \neq \phi$, then $\mathrm{pr}_\ell \rho_i = \underline{k}^{h-1}$ for $\ell = 1, \ldots, h$.*

Lemma 4.9. *No minimal polyrelation has arity greater than 4.*

We consider the quaternary minimal polyrelations. Put $\chi := \Delta_{12,34} \cup \Delta_{13,24} \cup \Delta_{14,23}$.

Lemma 4.10. *If $h = 4$, then each nonprimitive and nontrivial ρ_i contains χ.*

Proposition 4.11. *There is no minimal quaternary polyrelation.*

We consider ternary polyrelations.

Lemma 4.12. *Let $\lambda \in \Xi_1$ be ternary and such that $[\lambda]$ contains no proper binary polyrelation. Then*
$$\lambda_i \cap \iota_3 \in I := \{\phi, \omega_3, \Delta_{12}, \Delta_{13}, \Delta_{23}\}$$
for all $i \geq 0$.

6

Lemma 4.13. *If $h = 3$ each nontrivial ρ_i has $\nu_i = \omega_3$.*

We need the following result from [27] (explicitly in [29]). For an operation $f \in Q_k^{(n)}$ let f^o denote the $(n+1)$-ary relation
$$f^o := \{a_1 \ldots a_n f a_1 \ldots a_n \mid a_1, \ldots, a_n \in \underline{k}\}.$$

Lemma 4.14. *Let σ be a ternary relation and let $\sigma = \mu \cup \omega_3$ where $\phi \neq \mu \subseteq \sigma_3$. Suppose that every relation from $[\sigma]$ that is (i) at most binary, (ii) totally reflexive, or (iii) ternary of the form $\lambda \cup \Delta_{12}$ or $\lambda \cup \Delta_{12} \cup \Delta_{13}$ with $\lambda \subseteq \sigma_3$ is trivial. Then $[\sigma]$ contains m^o where $m(x_1, x_2, x_3) = x_1 - x_2 + x_3$ for all $x_1, x_2, x_3 \in \underline{k}$ and $<\underline{k}; +>$ is an abelian elementary p-group (p prime).*

Proposition 4.15. *There is no minimal ternary polyrelation.*

Summing up:

Theorem 4.16. *The set Ξ_3 of unary and binary polyrelations is B-generic.*

5. Unary Polyrelations

5.1 In this section we study the set U of proper unary polyrelations. Let $\rho \in U$. Let p^* denote the minimum of p_τ for unary, periodic and proper $\tau \in [\rho]$. Without loss of generality, we may assume that $p_\rho^* = p_\rho$, and denote this value by p. For a polyrelation λ set $c_\lambda := |\lambda_0| + \ldots + |\lambda_{p-1}|$. Clearly for λ unary c_λ takes on only finitely many values and therefore we can define
$$c_\rho^* := \min\{c_\lambda \mid \lambda \in [\rho], \lambda \text{ unary proper}\}.$$
Again without loss of generality, we may assume $c_\rho^* = c_\rho$ and write c for this common value. We have:

Lemma 5.2. *The sets $\rho_0, \ldots, \rho_{p-1}$ are pairwise disjoint.*

Lemma 5.3. *There is a divisor r of p and $T \subseteq \underline{r}$ such that $\rho_i \neq \phi$ if and only if $i \equiv t \pmod r$ for some $t \in T$. Moreover $\rho_i \neq \phi$ if and only if $i \equiv 0 \pmod r$.*

Lemma 5.4. *If $r = dr'$ and $d > 1$, then $(d, p) > 1$.*

Corollary 5.5. *Every prime divisor of r divides p.*

6. Binary Areflexive Polyrelations

6.1 In this section we consider binary polyrelations. A binary relation σ is *reflexive* (*areflexive*) if $\sigma \supseteq \iota_2$ ($\sigma \cap \iota_2 = \phi$). Let ρ be a binary polyrelation from Ξ_4 with period p. Setting $\theta = \{1,2\}^2$ in Lemma 4.6 we obtain that each $\rho_i \cap \iota_2$ is trivial, i.e. ρ_i is either reflexive or areflexive. Put
$$V_1 := \{i \in \underline{p} \mid \iota_2 \subset \rho_i \subset \underline{k}^2\},$$
$$V_2 := \{i \in \underline{p} \mid \rho_i \in \{\iota_2, \underline{k}^2\}\},$$
$$V := V_1 \cup V_2,$$
$$W_1 := \{i \in \underline{p} \mid \rho_i \neq \phi, \rho_i \cap \iota_2 = \phi\},$$
$$W_2 := \{i \in \underline{p} \mid \rho_i = \phi\},$$
$$W := W_1 \cup W_2,$$
$$\rho_v' := \rho_v \ (v \in V_1), \quad \rho_v' := \iota_2 \ (v \in V_2),$$
$$\rho_w'' := \rho_w \ (w \in W_1), \quad \rho_w'' := \phi \ (w \in W_2),$$
$$F := \text{Pol}\rho, \text{ and } D := \{d \in \underline{p} \mid F_d \neq \phi\}.$$

For $x, y \in Z$ let $x \dotplus y$ denote the element of $\underline{p}$ congruent $x + y$ modulo p. Further for $T \subseteq \underline{p}$ and $d \in \underline{p}$ put $d \dotplus T = \{d \dotplus t \mid t \in T\}$ and observe that for $r = (d, p)$ (greatest common divisor) the following conditions are equivalent: (i) $d \dotplus T \subseteq T$, (ii) $d \dotplus T = T$, (iii) $r \dotplus T = T$. We need the following:

Lemma 6.2. *We have $d \dotplus V = V$ and $d \dotplus W = W$ for each $d \in D$. If $V_1 \neq \phi$ ($W_1 \neq \phi$), then $\rho' \in [\rho]$ ($\rho'' \in [\rho]$) is proper.*

A binary polyrelation $\rho = (\rho_0, \rho_1, \ldots)$ is *areflexive* (*reflexive*) if each ρ_i is areflexive (reflexive and $\neq \underline{k}^2$). Let A and R denote the set of areflexive (reflexive) polyrelations from Ξ_4. We have:

Lemma 6.3. *The set $\Xi_5 := U' \cup A \cup R$ is B-generic.*

6.4 For $\rho \in A$ with period p put
$$I_\rho := \{i \in \underline{p} \mid \rho_i \neq \phi\}, \quad J_\rho := \underline{p} \backslash I_\rho,$$
and $j_\rho := |J_\rho|$.

For a map ψ from a subset D of $\underline{k}$ into $\underline{k}$ put $\psi^o := \{x\psi(x) \mid x \in D\}$. Let S denote the set of all permutations of $\underline{k}$. Finally let A' be the set of all $\rho \in A$ such that $\rho_i \in S^o := \{s^o \mid s \in S\}$ for all $i \in I_\rho$. We have:

Lemma 6.5. *The set $\Xi_6 := U' \cup A' \cup R$ is B-generic.*

6.6 We say that a polyrelation $\rho \in A'$ is *strict* if each nontrivial $\lambda \in [\rho] \cap A'$ has $p_\lambda = p_\rho$, $j_\lambda = j_\rho$ and $\text{Pol}\lambda = \text{Pol}\rho$ (i.e. we cannot improve on the period or number of empty components and $\text{Pol}\rho$ is not properly contained in another $\text{Pol}\lambda$). Let A'' be the set of strict polyrelations. We have:

Lemma 6.7. *The set $\Xi_7 := U' \cup A'' \cup R$ is B-generic.*

For a divisor q of k let S_q be the set of $s \in S$ with k/q cycles each of length q. Finally for $0 \leq j \leq p$ and q prime divisor of k let θ_{jp}^q

consist of $\rho \in A"$ with $p_\rho = p$, $j_\rho = j$ and $\rho_i \in S_q^{\,o}$ for all $i \in I_\rho$. Now we have:

Lemma 6.8.
$$A" \subseteq \bigcup_{q,j} \{\theta_{jp}^q \mid 0 \le j < p, \quad q \text{ prime divisor of } k\}.$$

6.9 We consider a fixed $\rho \in \theta_{jp}^q \cap A$. Without loss of generality we assume $0 \in I := I_\rho$.

Let $\rho_i = s_i^o$ for $i \in I$ and let G be the permutation group on $\underline{k}$ generated by $\{s_i \mid i \in I\}$. Further put $S' := \mathrm{id} \cup \bigcup_d \{S_d \mid d \text{ prime divisor of } k\}$. Now:

Lemma 6.10. *We have*
i) $G \subseteq S'$, and
ii) *if* $s = s_{i_1}^{\varepsilon_1} \ldots s_{i_n}^{\varepsilon_n}$ *for* $i_1, \ldots, i_n \in I$, $\varepsilon_1, \ldots, \varepsilon_n \in Z$, and
$$\nu_\ell := \rho_{\ell+i_1}^{\varepsilon_1} \circ \ldots \circ \rho_{\ell+i_n}^{\varepsilon_n}$$
for $\ell = 0, \ldots, p-1$, *then*
a) *all* $\nu_\ell \in \{\phi, \imath_2\}$ *if* $s = \mathrm{id}$, *and*
b) $\nu \in \theta_{jp}^d$ (*d prime divisor of* k) *otherwise*.

Lemma 6.11. *Let* $0 < d < p$ *and let* C_1 *and* C_2 *be the vertex sets of cycles of* ρ_i *and* ρ_{i+d}. *If* $|C_1 \cap C_2| > 1$, *then there are:* (i) *a divisor r of* p, (ii) $0 < m < q$, *and* (iii) $T \subseteq \underline{r}$ *such that*
a) $m^n \equiv 1 \pmod{q}$ *where* $n = p/r$,
b) $I = \{t + ir \mid t \in T, 0 \le i < n\}$,
c) $\rho_{t+ir} = \rho_t^{m^i}$ *for all* $t \in T$ *and* $0 \le i < m$.

At present we have only some very partial results in this direction and therefore for the time being we abandon areflexive polyrelations and turn to the reflexive ones.

7. Orders

7.1 In this section we consider binary reflexive polyrelations (all components reflexive). A binary relation σ is said to be *symmetric, antisymmetric* and *transitive* if $\sigma = \sigma^{-1}$, $\sigma \cap \sigma^{-1} = \imath_2$ and $\sigma^2 = \sigma$, respectively. An antisymmetric and transitive relation is an *order*. A polyrelation ρ has such a property if all ρ_i do. Let Rs and An denote the sets of proper periodic symmetric and antisymmetric binary polyrelations, respectively.

We start with the following.

Lemma 7.2. *The set* $\Xi_8 := U' \cup A" \cup An \cup Rs$ *is B-generic.*

7.3 In the remainder of this section we study An. Let P be the set of all $\rho \in An$ such that $\rho_i = \imath_2$ or

is a bounded order (i.e. there are $o_i, e_i \in \underline{k}$, called the least and greatest elements of ρ_i, such that $o_i \times \underline{k} \subseteq \rho_i$ and $\underline{k} \times e_i \subseteq \rho_i$). We have:

Lemma 7.4. i) *Every* $\rho \in An$ *is dominated by a polyrelation from* $P \cup Rs$.
ii) *Every transitive* $\rho \in An \backslash P$ *is dominated by a polyrelation from* Rs.

Let P' consist of all $\lambda \in P$ such that
(i) no polyrelation from $[\lambda] \cap Rs$ is proper,
(ii) each proper $\tau \in [\lambda] \cap P$ has $p_\tau \ge p$ and
$$|\tau_0| + \ldots + |\tau_{p-1}| \ge |\lambda_0| + \ldots + |\lambda_{p-1}|$$
where $p = p_\lambda$.
We have:

Lemma 7.5. *The set* $\Xi_9 := U' \cup A" \cup P' \cup Rs$ *is B-generic.*

Now we determine P'. In 7.6 – 7.9 ρ is a fixed polyrelation from P' with period p, $c := |\rho_0| + \ldots + |\rho_{p-1}|$, $I := \{i \in \underline{p} \mid \rho_i \supset \imath_2\}$ and $0 \in I$. We have:

Lemma 7.6. *We have* $\rho_i \cap \rho_{i+d} = \imath_2$ *for all* $0 < d < p$.

Lemma 7.7. *We have* $p = 2r$, $I = \{0, r\}$, *and* $\rho_r = \rho_0^{-1}$.

Lemma 7.8. *We have* $p = 2^m$ *for some* $m > 0$.

Summing up:

Theorem 7.9. *The set* P' *consists of* ρ *with period* 2^m ($m > 0$) *and such that* ρ_0 *is a bounded order,* $\rho_{2^{m-1}}$ *its converse and* $\rho_i = \imath_2$ *otherwise* ($i = 1, \ldots, 2^m-1$, $i \nmid 2^{m-1}$).

8. Reflexive Symmetric Polyrelations

8.1 In this section we study Rs.

Let σ be a binary reflexive and symmetric relation. A subset C of $\underline{k}$ is a *clique* of σ if $C^2 \subseteq \sigma$. The *center* C_σ of σ is the set $\{a \in \underline{k} \mid a \times \underline{k} \subseteq \rho\}$. If $\phi \ne C_\sigma \subseteq \underline{k}$ the relation σ is said to be *central*. Given binary relations σ_1 and σ_2 define
$$\sigma_1 * \sigma_2 := \{xy \mid xu, yv \in \sigma_1 \text{ and } uy, vx \in \sigma_2 \text{ for some } u, v \in \underline{k}\}.$$
It is easy to see that $\sigma_1 * \sigma_2$ is always symmetric and $\sigma_1 \cup \sigma_2 \subseteq \sigma_1 * \sigma_2$ provided both σ_1 and σ_2 are reflexive.

A binary polyrelation λ is *central* if each $\lambda_i \in \{\imath_2, \underline{k}\}$ or is central.

A polyrelation $\rho \in Rs$ with period p is *rich*, if every proper $\lambda \in [\rho] \cap Rs$ satisfies (i) $p_\lambda \ge p$,

(ii) $|\lambda_0|+\ldots+|\lambda_{p-1}| \leq |\rho_0|+\ldots+|\rho_{p-1}|$, and

(iii) $\text{Pol}\lambda = \text{Pol}\rho$. Let Rr be the set of rich polyrelations. We have:

Lemma 8.2. *The set* $\Xi_{10} := U' \cup A'' \cup P' \cup Rr$ *is B-generic.*

In the remainder of the paper we establish some properties of Rr. Let ρ be a fixed polyrelation from Rr with period p, and such that
$$\iota_2 \subset \rho_0 \subset \underline{k}^2.$$

For $I \subseteq \underline{p}$ set $\rho_I := {}_{i\in I}\cap\rho_i$. Further let $\mathcal{J}$ consist of nonempty subsets I of $\underline{p}$ such that
$$\rho_I \supset \iota_2.$$

An m-ary operation u on $\underline{k}$ is *idempotent* if $ux\ldots x = x$ for all $x \in \underline{k}$. We have:

Lemma 8.3. *Let* $0 < m < p$ *and* $I_0=\{0\},I_1,\ldots,$ $I_{m-1} \subseteq \underline{p}$ *such that* $0 \notin I_i \in \mathcal{J}$ *for some* $0 < i < m$. *Then there exists an m-ary idempotent operation u on $\underline{k}$ such that*
$$(ux_0\ldots x_{m-1}, ux_0\ldots x_{i-1}yx_{i+1}\ldots x_{m-1}) \in \rho_{z+I_\ell}$$
for all $x_0,\ldots,x_{m-1}, y \in \underline{k}$, $\ell \in \underline{m}$ *and* $z \in \underline{p}$.

We say that ρ is *complementing* if (i) $\mathcal{J}$ contains all singletons $\{i\}$ ($i \in \underline{p}$) but not $\underline{p}$, and (ii) if $0 \notin I \in \mathcal{J}$, and $z \in \underline{p}$, then there are (a) a cover of $\underline{k}$ by cliques $C_1, \ldots, C_n$ of ρ_z and (b) a cover of $\underline{k}$ by cliques $D_1, \ldots, D_m$ of ρ_{z+I} such that each C_p meets every D_q ($1 \leq p \leq n, 1 \leq q \leq m$).

Lemma 8.4. *The polyrelation ρ is complementing. The relations $\rho_0, \ldots, \rho_{p-1}$ are pairwise distinct and distinct from ι_2. For $(i, j) \in \underline{p}^2$, $i \neq j$, there are covers of $\underline{k}$ by cliques $C_1, \ldots, C_n$ and $D_1, \ldots, D_m$ of ρ_i and ρ_j, respectively, such that each C_q meets every D_r ($1 \leq q \leq n$, $1 \leq r \leq m$).*

Lemma 8.5. *The relations $\rho_0, \ldots, \rho_{p-1}$ are either (i) all equivalences $\neq \iota_2$ or (ii) central or equal $\underline{k}^2$.*

Bibliography

[1] R. A. Bairamov, On the question of functional completeness in many-valued logics (Russian), Diskret. Analiz, 11 (1967), 3-20. MR 33 # 5712

[2] L. A. Birjukova and V. B. Kudrjavcev, On completeness of functions with delays (Russian), Problemy Kibernet., 23 (1970), 5-25. English translation: Systems Theory Research, 23 (1973), 3-24. MR 45 # 39

[3] J. Dassow, Completeness Problems in the Structural Theory of Automata, Akademie-Verlag, Berlin, 1981.

[4] T. Hikita, Completeness criterion for functions with delay defined over a domain of three elements, Proc. Japan Acad., 54 (1978), 335-339.

[5] T. Hikita, Completeness properties of k-valued functions with delays: Inclusions among closed spectra, Math. Nachr., 103 (1981), 5-19.

[6] T. Hikita, On completeness for k-valued functions with delay, Coll. Math. Soc. János Bolyai 28, Finite Algebra and Multiple-Valued Logic (B. Csákány and I. Rosenberg eds.), pp. 345-371, North-Holland, 1981.

[7] T. Hikita and A. Nozaki, A completeness criterion for spectra, SIAM J. Comput., 6 (1977), 285-297. Corrigenda, ibid., 8 (1979), 656.

[8] S. V. Jablonskii, Functional constructions in the k-valued logic (Russian), Trudy Mat. Inst. Steklov., 51 (1958), 5-142. MR 21 # 3331

[9] R. E. Kricevskii, The realization of functions by superpositions (Russian), Problemy Kibernet., 2 (1959), 123-138. German translation: Probleme der Kybernetik, 2 (1963), 139-159.

[10] V. B. Kudrjavcev, Completeness theorem for a class of automata without feedback couplings (Russian), Dokl. Akad. Nauk SSSR, 132 (1960), 272-274. English translation: Soviet Math. Dokl., 1 (1960), 537-539.

[11] V. B. Kudrjavcev, Completeness theorem for a class of automata without feedback couplings (Russian), Problemy Kibernet., 8 (1962), 91-115. German translation: Probleme der Kybernetik, 8 (1965), 105-136. MR 30 # 28.

[12] V. B. Kudrjavcev, The power of sets of pre-complete sets for certain functional systems connected with automata (Russian), Problemy Kibernet., 13 (1965), 45-74. MR 35 # 6493.

[13] V. B. Kudrjavcev, On functional properties of logical nets (Russian), Math. Nachr., 55 (1973), 187-211.

[14] V. B. Kudrjavcev, Functional Systems (Russian), I.M.U., Moscow, 1982.

[15] A. V. Kuznecov, Forty years of Soviet mathematics (Russian), T. I., pp. 102-115, Fizmatgiz, Moscow, 1957.

[16] A. V. Kuznecov, Structures with closures and criteria of functional completeness (Russian), Uspechi Mat. Nauk, 16(98) (1961), 201-202.

[17] H. H. Loomis, Jr., Completeness of sets of delayed-logic devices, IEEE Trans. Electron. Comput., EC-14 (1965), 157-172.

[18] L. Martin, C. Reischer and I. G. Rosenberg, Completeness problems for switching circuits constructed from delayed gates, Proc. 8th Int. Symp. Multiple-Valued Logic, pp. 142-148, 1978. To appear in Elektron. Informationsverarb. Kybernet. (in French). An expanded French version appeared in Reseaux modulaires, Monographies de 1.U.Q.T.R. no 11, 91p., 1980.

[19] A. Nozaki, Réalisation des fonctions definies dans un ensemble fini à l'aide des organes élémentaires d'entrée-sortie, Proc. Japan Acad., 46 (1970), 478-482.

[20] A. Nozaki, Complete sets of switching ele-
     ments and related topics, First USA-Japan
     Computer Conference, pp. 393-396, 1972.
[21] A. Nozaki, Functional completeness of multi-
     valued logical functions under uniform
     compositions, Rep. Fac. Eng. Yamanashi Univ.,
     29 (1978), 61-67.
[22] A. Nozaki, Completeness criteria for a set
     of delayed functions with or without non-
     uniform compositions, Coll. Math. Soc. János
     Bolyai 28, Finite Algebra and Multiple-
     Valued Logic (B. Csákány and I. Rosenberg
     eds.), pp. 489-519, North-Holland, 1981.
[23] A. Nozaki, Maximal *-incomplete sets of func-
     tions defined over the set {0,1,2}, preprint.
[24] R. Pöschel and L. A. Kaluznin, Funktionen-
     und Relationenalgebren, VEB Deutscher Verlag
     der Wissenschaften, Berlin, 1979.
[25] E. L. Post, Introduction to a general theory
     of elementary propositions, Amer. J. Math.,
     43 (1921), 163-185.
[26] I. G. Rosenberg, La structure des fonctions
     de plusieurs variables sur un ensemble fini,
     C. R. Acad. Sci. Paris Sér. A-B, 260 (1965),
     3817-3819. MR 31 # 1185
[27] I. G. Rosenberg, Über die funktionale
     Vollständigkeit in dem mehrwertigen Logiken
     (Struktur der Funktionen von mehreren
     Veränderlichen auf endlichen Mengen),
     Rozpravy Ceskoslovenské Akad. Ved., Ser.
     Math. Nat. Sci., 80 (1970), 3-93. MR 45
     # 1732
[28] I. G. Rosenberg, Completeness properties of
     multiple-valued logic algebras, in "Computer
     Science and Multiple-Valued Logic, Theory and
     Applications" (D. C. Rine ed.), pp. 144-186,
     North-Holland, 1977.
[29] I. G. Rosenberg and L. Szabó, Local complete-
     ness I, preprint CRMA-1072 (Nov. 1981), to
     appear in Algebra Universalis.

Session 1A
Logic Design I

A STUDY OF REDUCED DEPENDENCE IN MULTI-VALUED SEQUENTIAL MACHINES

T. C. Yang and A. S. Wojcik

Department of Information and
Computer Engineering
Feng Chia University
Taichung, Taiwan

Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois 60540
U. S. A.

ABSTRACT

The problem of determining the exist-
ence of state assignments for multi-valued
sequential machines with the property of
reduced dependence is considered. The
properties of and requirements for reduced
dependence among state variables in the
binary system are extended to that for
multi-valued synchronous and asynchronous
machines. A number of examples to illus-
trate these concepts are also presented.

1. INTRODUCTION

Even though there have been many studies
on non-binary logic design, [1] - [5],
and multi-valued circuit implementations,
[6] - [8], there are still a number of
design problems, the solutions to which
would enhance the potential usefulness of
multi-valued systems. One such problem
which has received considerable attention
in the binary sequential case is that of
determining the existence of state assign-
ments with the property of reduced depend-
ence, that is, the next state equations of
the state variables are independent of some
of the state variables. Hartmanis [9]
first developed the basic tool for the
analysis of reduced dependence, namely the
partition having the substitution property
on the set of states of a sequential ma-
chine. Stearns and Hartmanis [10], Hart-
manis and Stearns [11, 12] then established
the concepts of partition and pair algebras
and applied them to the problem of designing
sequential circuits with reduced dependence.
Weiner and Smith [13] describe an algorithmic
"solution" based upon applying the partition
algebra to the state assignment problem for
synchronous sequential machines. The
algorithm can assign the input, state and
output variables of a given machine so as
to minimize the total logic, that is,
reduced dependencies of both the state and
output logic on state and input variables
are optimized. Tan, Menon, and Friedman
[14] extended the theories of reduced
dependence developed by Hartmanis and

Stearns [12] for synchronous sequential
machines to asynchronous sequential ma-
chines. This paper is concerned with ex-
tending the existing concepts of partition
systems and substitution properties to
study the properties and requirements of
reduced dependence among state variable
in multi-valued synchronous and asynchro-
nous sequential machines. Section II dis-
cusses the physical properties which multi-
valued devices must exhibit. Assumptions
concerning delays, transitions, and actual
device reactions are presented, and an
operational model is described. Section
III presents the properties of partitions
and their application to state assignments.
The necessary and sufficient conditions
for reduced independence between states
are established. Since the 3-valued system
is the easiest to describe, this system is
primarily used in examples and in the
proofs of certain theorems. However, all
the concepts to be presented are directly
applicable (with either no modification or
just straightforward extensions) to arbi-
trary multi-valued systems.

2. NOTATION AND BASIC OPERATIONAL CONCEPTS

Let us denote the R logic values in a R-
valued system as $0,1,2,...,r-1$. Defining
the Postian operations of addition $(+)$ and
multiplication $(.)$ on the set R as

$$Y + X = X + Y = MAX\ (X,Y)$$

$$Y \cdot X = X \cdot Y = XY = MIN\ (X,Y)$$

Yields a distributive lattice with zero
element 0 and universal element $r-1$.
Let us define unary operators as follows:

$$
{}^{i}_{r-1}{}^{i}_{r-2} \cdots {}^{i}_{2}{}^{i}_{1}{}^{i}_{0} =
\begin{cases}
i_0 & \text{if } X = 0 \\
i_1 & \text{if } X = 1 \\
i_2 & \text{if } X = 2 \\
\vdots & \vdots \\
i_{r-2} & \text{if } X = r-2 \\
i_{r-1} & \text{if } X = r-1
\end{cases}
$$

12

where $i_j \in \{o,1,2,\ldots,r-1\}$ for $0 < j < r-1$.

Three different orderings can be imposed on the r values associated with a R-valued system. If transferring from value i to value j requires that all intermediate values between i and j be attained, the ordering is said to be <u>linear</u>. If the transition from r-1 to 0 can be made directly, the ordering is said to be <u>rotational</u> or <u>cyclic</u>. If it is possible to transfer directly from any value i to any value j, the ordering is said to be <u>complete</u>.

Since any design using linearly ordered variables will work for other orderings, it will be assumed, in this paper, that all variables can take on only linearly ordered values.

The assumption of linear ordering places constraints on transitions which may occur within a circuit. There is the question of whether or not the circuit will react to the intermediate values which must occur during the during the transition. Two major assumptions are made in this paper concerning this situation:

A. Line Reaction Assumption:

If the value on a given line in the circuit is changed from i to j, it will take on all intermediate values between i and j in linear fashion from i to j. No assumptions are made about how fast such a transition will occur. For example, let us consider a circuit with two 3-valued variables X_1 and X_2. Suppose that the values are changing from $X_1 X_2 = 00$ to $X_1 X_2 = 22$. The possible paths of transition are 00-10-20-21-22, 00-10-11-21-22, 00-10-11-12-22, 00-01-11-12-22, 00-01-02-12-22, and 00-01-11-21-22.

B. Gate Reaction Assumption:

When an input to a logical device changes from i to j, it will assume every value in order from i to j. Each such value may exist as the input for some arbitrary finite length of time and, furthermore, the circuit may or may not react to each of these inputs. If the reaction calls for an output change, that change will proceed in linear order, with each value being assumed for some finite length of time. As an example, let us consider the function given in Figure 1. This is a single-input, single-output four-valued function. Suppose first that X=0, so that f(X)=2. Now, if X changes to 3, the following are three possible sequences of values for f(X):

    1. f reacts to all input values, so f(X) takes on values in the sequence 2,1,2,3,2,1,0.

2. f reacts to X = 1, but not X = 2. The sequence of values for f(X) is 2,1,0.

3. f does not react to X = 1 or 2. The sequence of values for f(X) is 2,1,0.

The above assumptions have placed as few restrictions as possible on the physical devices, so that circuits designed to work under these assumptions will work under a wide range of assumptions.

3. REDUCED DEPENDENCE IN MULTI-VALUED SEQUENTIAL MACHINES

3.1 Partition Systems

Hartmanis and Stearns [12] first developed the partition and pair algebras and applied these concepts to sequential circuit design. In this section, we shall first present the essential definitions and notation based on their work. It should be pointed out that the concepts to be described are directly applicable to an arbitrary R-valued machine.

<u>Definition 3.1</u> A <u>sequential</u> <u>machine</u> is a quintuple

$$M = (S,I,O,N,Z)$$

where   S is the set of states, I is the set of inputs,

    O is the set of ouputs,

    N : S x I → S is called the next state function, and

    Z : S x I → O is called the output function.

A general model of a sequential switching circuit is given in Figure 2. In this model, the (primary) input variables are denoted by $i_1, \ldots, i_t$. The present-state variables or secondary input variables are denoted by $y_1, \ldots, y_p$; the (primary) output variables are noted by $z_1, \ldots, z_s$; and the next-state variables or secondary output variables are denoted by $Y_1, \ldots, Y_p$.

<u>Definition 3.2</u> A <u>Partition</u>, π, on a set of states, S, is a collection of subsets $B_1, B_2, \ldots, B_n$ such that $\bigcup_n B_n = S$ and $B_i \cap B_j = \emptyset$ if $i \neq j$ where $\emptyset$ is the empty set. $B_1, B_2, \ldots, B_n$ are called <u>blocks</u> of the partition.

If the states s_i and s_j are in the same block of π, this will be denoted by $s_i = s_j (\pi)$.

<u>Definition 3.3</u> The ordered pair of

13

partitions (π_1, π_2) defined on S is a __partition pair__ for M, denoted by $P(\pi_1, \pi_2)$, if and only if for each block B_i of π_1 and each input I_m, there exists a block B_j of π_2 such that $N(B_i, I_m) \subseteq B_j$.

__Definition 3.4__ A partition π defined on S is said to have the __substitution property__, denoted by SP (π), if and only if $P(\pi, \pi)$. If a partition has the substitution property, then it is called a __SP Partition__.

__Definition 3.5__ For two partitions π_1 and π_2 defined on the same set, π_1 is __smaller than or equal to__ π_2, denoted by $\pi_1 \leq \pi_2$, if and only if every block of π_1 is contained in a block of π_2.

__Definition 3.6__ The __product__ of two partitions π_1 and π_2, denoted by $\pi_1 \cdot \pi_2$, defined on the same set is a partition such that

a) $\pi_1 \cdot \pi_2 \leq \pi_1$
b) $\pi_1 \cdot \pi_2 \leq \pi_2$
c) if for any other partition π_i, $\pi_i \leq \pi_1$ and $\pi_i \leq \pi_2$, then $\pi_i \leq \pi_1 \cdot \pi_2$.

The product of n partitions $\pi_1, \pi_2, \ldots, \pi_n$ is denoted by $\prod_{i=1}^{n} \pi_n$.
In order to incorporate the features of R-valued logic, the following definitions and notation are presented to allow variables and functional values to be associated with a set of values.

3.2 Unicovered State Assignments

__Definition 3.7__ An __n-block partition__ is a partition in which there are exactly n blocks.

__Definition 3.8__ Given $a \leq b$ in a poset A, the __interval__ $[a,b]$ is defined to be the set of all $x \in A$ such that $a \leq x \leq b$. Since $R = \{0,1,2,\ldots r-1\}$ is a poset under the relationship of "less than or equal to", $\leq$, $[2]$, given $a_i \leq a_j$ in R, the interval $[a_i, a_j]$ is defined to be the set of all $a_t \in R$ such that $a_i \leq a_t \leq a_j$, that is, the ordered sequence $a_i, a_{i+1}, \ldots, a_{j-1}, a_j$.

__Definition 3.9__ Let $R = \{0,1,2,\ldots,r-1\}$ be a set of non-negative integers. Define a __sequential partition__ of R as

$$\hat{R} = (B_1; B_2; \ldots ; B_t)$$

where $B_i \cap B_j = 0$ for $1 \leq i \neq j \leq t$

$B_1 \cup B_2 \ldots \cup B_t = [0, r-1] = 0,1,2,\ldots,r-1$

and

$B_1 = [0, a_1] = 0,1,2,\ldots,a_1$
.
.
.
$B_t = [a_{t-1}+1, r-1] = a_{t-1}+1, a_{t-1}+2, \ldots, r-1$

where $0 \leq a_i \leq r-1$ for $1 \leq i \leq r-1$

__Definition 3.10__ Let $\pi = (B_1, B_2, \ldots, B_n)$ be an n-block partition defined on the set of states S. An R-valued y-variable __covers__ the n-block partition, denoted by τ_y, if y is assigned the values

$0,1,2,\ldots,a_{i_1}$ for states in B_{i_1}

$a_{i_1} + 1, a_{i_1} + 2, \ldots, a_{i_2}$ for states in B_{i_2}
.
.
$a_{i_{n-1}} + 1, a_{i_{n-1}} + 2, \ldots, r-1$ for states in B_{i_n}

where $0 \leq a_{i_j} \leq r-1$ for $1 \leq i_j \leq n-1$

and B_{i_j} is a block in $(B_1, B_2, \ldots, B_n)$ for $1 \leq i_j \leq n$.

In the above definition, note that the intervals for the B_{i_j} form a sequential partition on R. Since each element in the set of states S appears in one and only one block of the partition, the partition is said to be __unicovered__ by a y-variable. If every y-variable in a state assignment covers a partition, then the assignment is called an __unicovered state assignment__. Let us use an example to illustrate the proceeding definitions.
Example 3.1 In the ternary case, $R = \{0,1,2\}$. Let us consider the case of a 2-block partition first. Assume that $(B_1, B_2) = (123, 456)$, then $\tau_y = (123,456)$ has the four possible distinct coverings as given in Table I. If we have a 3-block partition $(B_1, B_2, B_3) = (12,34,56)$, then $\tau_y = (12,34,56)$ will have the six possible distinct coverings as shown in Table II.

__Definition 3.11__ Let $\hat{R}$ be a sequential partition on $\hat{R}$ such that $\hat{R} = (B_1; B_2; \ldots; B_t)$.

Define a <u>sequential</u> <u>partition</u> <u>function</u> F such that

$$F(B_i) = B_j \text{ for } 1 \leq i, j \leq t.$$

In other words, a sequential partition is really an ordered sequence of intervals, and a sequential partition function can map intervals to intervals.

As an example, if $R = \{0,1,2\}$ and $\hat{R} = (0; 1,2) = (B_1; B_2)$, then we may define $F(B_1) = F(0) = B_2 = 1,2;$ and $F(B_2) = F(1,2) = B_1 = 0.$

A <u>valid</u> unicovered state assignment in variables $y_1, y_2, \ldots, y_n$ on a set of states S generates a set of partitions $\tau_{y_1}, \tau_{y_2}, \ldots, \tau_{y_n}$ such that $\prod_{i=1}^{n} \tau_{y_i} = \emptyset$. If it is the case that $\prod_{i=1}^{n} \tau_{y_i} \neq \emptyset$, then there must be two states s_i and s_j such that

$$s_i = s_j \ (\tau_{y_k}) \text{ for all } 1 \leq k \leq n,$$

and hence s_i and s_j have the same coding $y_1, y_2, \ldots, y_n$, that is, they are indistinguishable, and, by definition, the state assignment is not considered to be valid.

In the next two sections, we shall discuss state assignments with reduced dependence and their relation to partition systems. Our discussion will cover both the synchronous and the asynchronous cases.

3.3 Reduced Dependence in Multi-Valued Synchronous Machines

In binary systems, the complexity of the circuit as well as its structural properties are strongly dependent upon the state codes chosen. The same situation is also true in R-valued systems. In this section, the property of reduced dependence in multi-valued synchronous sequential machines, in the sense of some of the state variables, will be discussed. We assume that clocked D-type flip-flops are used as memory elements in the following analysis.

The following theorems are based on similar results for binary systems $\lfloor 15 \rfloor$:

<u>Theorem 3.1</u> Given a valid unicovered state assignment in variables $y_1, y_2, \ldots, y_n$ on the set of states $S = \{ s_1, s_2, s_3, \ldots, s_t \}$ of a machine M, Y_k, the next state equation of y_k, is independent of y_j if and only if $P(\prod_{i \neq j} \tau_{y_i}, \tau_{y_k})$. (See Appendix A for a proof)

<u>Theorem 3.2</u> Given a valid unicovered state assignment in variables $y_1, y_2, \ldots, y_n$ on the states S of a machine M, Y_k, the next state equation of y_k, is dependent on a proper subset S_i of S, if and only if $P(\prod_{y_i \in S_i} \tau_{y_i}, \tau_{y_k})$.

(See Appendix B for a proof).

Note that in Theorem 3.2, Y_k is dependent only on y_k if and only if $P(\tau_{y_k}, \tau_{y_k})$, that is, $SP(\tau_{y_k})$.

By applying Theorems 3.1 and 3.2, we can now consider a ternary example to demonstrate the property of reduced dependence.

Example 3.2 The state table for this example is given in Table III. $\prod_1 = (123,456)$ and $\prod_2 = (14,25,36)$ are two SP partitions of the states. If ternary variables y_1 and y_2 are now assigned to cover $\prod_1$ and $\prod_2$, respectively, we get $\tau_{y_1} = (123,456)$ and $\tau_{y_2} = (14,25, 36)$. Since $\tau_{y_1} \cdot \tau_{y_2} = \emptyset$, a valid unicovered state assignment can be based on τ_{y_1} and τ_{y_2}, as shown in Table IV. For y_1, the values 0, 1 and 2 are assigned for states 1 and 4, 2 and 5, 3 and 6, respectively. Since $P(\tau_{y_1}, \tau_{y_1})$, we predict, by Theorem 3.1, that Y_1 is independent of Y_2. Similarly, since $P(\tau_{y_2}, \tau_{y_2})$, we predict, by Theorem 3.2, that Y_2 is dependent only on y_2 (and the inputs). The corresponding transition table is given in Table IV. Base on the transition table, we derive the next state functions of y_1 and y_2:

$$Y_1 = y_1^{210} \cdot I_0 + F(B_1^{y1}) \cdot I_1$$
$$Y_2 = y_2^{101} \cdot I_0 + y_2^{102} \cdot I_1 + y_2^{102} \cdot I_2$$

where $B_1^{y1} = B_1 = 0$ for $y_1 = 0$, $B_1^{y1} = B_2 = 1,2$ for $y_1 = 1,2$; and as defined previously, $\hat{R} = (0; 1,2) = (B_1; B_2)$, $F(B_1) = F(0) = B_2 = 1,2$, and $F(B_2) = F(1,2) = B_1 = 0$.

Obviously, Y_1 and Y_2 fulfill our predictions. Since there are nine cells of Y_1 that allow multiple values 1 and 2, a total of 2^9 ways are possible for the implementation of the next state functions. The transition

table for one such implementation is shown in Tables V. Based on this table, we derive the next state functions of Y_1 and Y_2 as follows:

$$Y_1 = Y_1^{210} \cdot I_0 + y_1^{001} \cdot I_1$$

$$Y_2 = y_2^{101} \cdot I_0 + y_2^{102} \cdot I_1 + y_2^{012} \cdot I_2$$

In the following section, we will extend the sufficient and necessary conditions of reduced dependence discussed in Theorems 3.1 and 3.2 to asynchronous sequential circuits.

3.4 Reduced Dependence of Multi-Valued Asynchronous Machines

The necessary and sufficient conditions for a synchronous state assignment with reduced dependence is necessary but not sufficient for an asynchronous state assignment with reduced dependence because of the complications resulting from possible critical races. The reason is that in the synchronous case, all codes in the state variables which are not assigned to a state can be left unspecified. However, in the asynchronous case, it may be necessary to specify such codings to ensure proper operation. In the following discussion, we assume that delay lines (or D-type flip-flops) are used as memory elements in the asynchronous case. As in the binary case, a convenient means of representing a multi-valued asynchronous sequential machine is a flow table. Let us consider Table VI as an example. Suppose the initial total state is $(X,0)$, and the input I changes from 0 to 2. One would expect the final stable state to be Z, but this is not the case. For if the machine reacts to the 1 input which must occur between 0 and 2, it can change to stable state Y and when the 2 input is seen, the machine will remain in state Y which is an incorrect final stable state. This type of problem will occur whenever a transition is made from a stable state in an input column to another input column which differs from the first in exactly one variable but is not adjacent to it, that is, for single-input multiple-value input changes. In order to insure correct operation, one of the following conditions must exist:

1. Single-input multiple-value changes are not allowed at those places where transitions may result in incorrect final stable states.

2. The input variable may change, but, in the flow table, one of the following two conditions holds for each entry in the row which is under a column between the initial stable entry and the final entry.
   2.a. The entry is stable.

   2.b. The entry is a transition to the same row as the final entry, and the entry in that row is stable.

Definition 3.12 The <u>ordered n-chotomy</u> $(A_1, A_2, \ldots, A_n)$ (hereafter called an n-chotomy) is a collection of n sets of states $A_1, A_2, \ldots, A_n$, called the <u>first set</u>, <u>second set</u>,..., and <u>n-th set</u>, respectively such that $A_i \cap A_j = 0$ for $1 \leq i \neq j \leq n$. In the above definition, when n = 2, we have the standard dichotomy and when n = 3 we have the trichotomy.

Definition 3.13 A state assignment in which all the state variables are allowed to change simultaneously without critical races is called a <u>single transition time</u> (STT) <u>assignment</u>. Furthermore, if only a single coding is associated with each state, it is called a <u>unicode single transition time assignment</u>. In the multi-valued case, if a state assignment is an STT assignment and is also a unicovered state assignment, it is called a <u>unicovered single transition time</u> (U_n STT) <u>assignment</u>.

Definition 3.14 A ternary y-variable <u>covers</u> the dichotomy (A_1, A_2) if that variable is assigned the value(s) 0; 0,1;2; or 1,2 for states in the first set A_1 and 1,2; 2; 0,1; or 0, respectively for states in the second set A_2.

Definition 3.15 The trichotomy (A_1, A_2, A_3) is said to <u>cover</u> the dichotomy (A_1', A_2'), if either

1. ∃ A partition of A_1' into A_{11}' and A_{12}' such that

   a. $A_{11}' \subset A_1$, $A_{12}' \subset A_2$, and $A_2' \subset A_3$, or

   b. $A_{11}' \subset A_3$. $A_{12}' \subset A_2$, and $A_2' \subset A_1$,

   or
2. ∃ A partition of A_2' into A_{21}' and A_{22}' such that

   a. $A_1' \subset A_1$, $A_{21}' \subset A_2$, and $A_{22}' \subset A_3$, or

   b. $A_1' \subset A_3$, $A_{21}' \subset A_2$, and $A_{22}' \subset A_1$

Theorem 3.4 If the trichotomy (A_1, A_2, A_3) covers a dichotomy, and a ternary y-variable is assigned values such that it takes on the value 0 for states in A_1 (A_3), 1 for states in A_2 and 2 for states in A_3 (A_1), then that y-variable covers the dichotomy, [3].

In order to derive the necessary and sufficient conditions for reduced dependence in multi-valued asynchronous sequential circuits, the following definition is very

important.

Definition 3.17 A dichotomy $(s_i s_j, s_m s_n)$ associated with the transitions $s_i \to s_j$ and $s_m \to s_n$ is said to be *relevant to a partition* Π if and only if $s_j \neq s_n$ (Π).

The following theorem is based on a similar result for the binary case [15].

Theorem 3.5 Given a unicovered single transition time state assignment in variables $y_1, y_2, \ldots, y_p$ for a flow table M, let δ be a subset of the state variables. Then for any state variable y_j, $1 \leq j \leq p$, the next state equation is of the form $Y_j = f_j$ (δ, I_k), where I_k is the input vector, if and only if

    a. $P(\Pi_{y_i \in \delta} \ \tau_{y_i}, \tau_{y_j})$,

    b. Every dichotomy which is relevant to the partition τ_{y_j} is covered by some $y_i \not\in \delta$. (See Appendix C for a proof of this theorem)

4. CONCLUSION

The problem of determining the existence of state assignments for multi-valued sequential machines with the property of reduced dependence has been considered. The properties of, and requirements for, reduced dependence among state variables in the binary system were extended to that for multi-valued synchronous and asynchronous machines which use unicovered state assignments. [18] also considers the cases of nonunicovered state assignments. Other problems need to be considered. In determining the reduced dependence among state variable, it was assumed that delay lines or D-type flip-flops were used as memory elements. Analysis using other types of multi-valued memory elements such as multi-valued JK flip-flops [17] needs to be considered. Further work is also needed in the areas of extending the analysis of reduced dependence of a set of state variables on another set of state variables on the inputs, and the reduced dependence of outputs on state variables or inputs.

One of the most important aspects emphasized in this paper is that existing results concerning binary sequential machines can be extended to multi-valued ones. This close analogy between binary and non-binary systems should continue to be utilized by researchers.

Figure 1. A sample 4-Valued Function.

X	f(X)
0	2
1	1
2	3
3	0
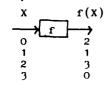
Figure 2. General Model of a Sequential Switching Circuit.
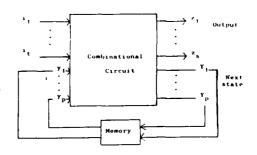
Table I. Assignment table for $\tau_y = (123, 456)$

123	456
0	1,2
0,1	2
2	0,1
1,2	0

Table II. Assignment table for $\tau_y = (12, 34, 56)$

12	34	56
0	1	2
0	2	1
1	0	2
1	2	0
2	0	1
2	1	0

Table III. State Table of Example 3.2

State \ Input	I_0	I_1	I_2
1	2	6	3
2	1	4	2
3	2	5	1
4	5	3	3
5	4	1	2
6	5	2	1

17
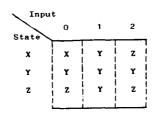
Table IV. Full Transition Table of Example 3.2

Present State	State Assignment		I_0		I_1		I_2	
	y_1	y_2	y_1	y_2	y_1	y_2	y_1	y_2
1	0	0	0	1	1,2	2	0	2
2	0	1	0	0	1,2	0	0	1
3	0	2	0	1	1,2	1	0	0
4	1,2	0	1,2	1	0	2	0	2
5	1,2	1	1,2	0	0	0	0	1
6	1,2	2	1,2	1	0	1	0	0

Table V. A Transition Table of Example 3.2

Present State	State Assignment		I_0		I_1		I_2	
	y_1	y_2	y_1	y_2	y_1	y_2	y_1	y_2
1	0	0	0	1	1	2	0	2
2	0	1	0	0	1	0	0	1
3	0	2	0	1	1	1	1	0
4	1	0	1	1	0	2	0	2
5	1	1	1	0	0	0	0	1
6	1	2	1	1	0	1	0	0
4	2	0	2	1	0	2	0	2
5	2	1	2	0	0	0	0	1
6	2	2	2	1	0	1	0	0

Table VI. A Ternary Flow Table

State \ Input	0	1	2
X	X	Y	Z
Y	Y	Y	Y
Z	Z	Y	Z

Appendix A: Proof for Theorem 3.1

Necessity: Suppose Y_k is independent of y_j and it is false that $P(\prod_{i\neq j} \tau_{y_i}, \tau_{y_k})$.

Let $N(s_i, I_m) = s_i' = (v_{i_1}, v_{i_2}, \ldots, v_{i_k}, \ldots, v_{i_n})$ where $s_i, s_i' \in S$. I_m is an input, and v_{i_k} is the value assigned to y_k in the state assignment of s_i'. Since it is false that $P(\prod_{i\neq j} \tau_{y_i}, \tau_{y_k})$, there must exist two states s_p and s_q and an input I_m such that $s_p = s_q(\tau_{y_i})$ for all $i\neq j$ and $N(s_p, I_m) \neq N(s_q, I_m)$ (τ_{y_k}). If these two states s_p and s_q do

not exist, then $\prod_{i\neq j} \tau_{y_i} = (s_1, s_2, \ldots, s_t)$ implies that y_j is a redundant state variable since the assignment in variables y_1, $y_2, \ldots, y_i, \ldots, y_{j-1}, y_{j+1}, \ldots, y_n$ is a valid state assignment. Based on τ_{y_k}, let us consider $N(s_p, I_m) = v_{p_k}$ and $N(s_q, I_m) = v_{q_k}$. Since $N(s_p, I_m) \neq N(s_q, I_m)(\tau_{y_k})$, $v_{p_k} \neq v_{q_k}$. If the next state function is represented in the sum of products form, then y_k will contain terms of the form $I_m \cdot v_{p_k} \cdot f(y_i)$. y_j^p and $I_m \cdot v_{q_k} \cdot f(y_i) \cdot y_j^q$, where $f(y_i)$ represents the common state assignment for s_p and s_q in variables y_i for all $i\neq j$, y_j^p and y_j^q represent the assignments in variable y_j for states s_p and s_q, respectively. Since $v_{p_k} \neq v_{q_k}$, the preceeding two product terms cannot be combined to eliminate y_j^p and y_j^q. Therefore Y_k is a function of y_j contradicting the original assumption that it was independent of y_j.

Sufficiency: Suppose $P(\prod_{i\neq j} \tau_{y_i}, \tau_{y_k})$. Then for any two states s_p and s_q such that $s_p = s_q(\tau_{y_i})$ for all $i\neq j$, $N(s_p, I_m) = N(s_q, I_m)$ (τ_{y_k}) for all I_m. Based on τ_{y_k}, let us consider $N(s_p, I_m) = v_{p_k}$ and $N(s_q, I_m) = v_{q_k}$. Since $N(s_p, I_m) = N(s_q, I_m)$ (τ_{y_k}) for all I_m, $v_{p_k} = v_{q_k}$ for all I_m. If the next state function is represented in the sum of products form, then Y_k will contain terms of the form $I_m \cdot v_{p_k} \cdot f(y_i) \cdot y_j^p$ and $I_m \cdot v_{q_k} \cdot f(y_i) \cdot y_j^q$, where $f(y_i)$, y_j^p, and y_j^q have the same meaning as defined previously. Since $v_{p_k} = v_{q_k}$, these product terms can be conbined as $I_m \cdot v_{p_k} \cdot f(y_i) \cdot (y_j^p + y_j^q)$.

Let us now examine $(y_j^p + y_j^q)$ in the ternary case. Since s_p and s_q are two distinct states, $y_j^p \neq y_j^q$, $y_j^p \not\subset y_j^q$, and $y_j^q \not\subset y_j^p$. We also know that $y_j^p, y_j^q \in \{y_j^{002}, y_j^{020}, y_j^{200}, y_j^{220}, y_j^{022}\}$. If $(y_j^p + y_j^q) = (y_j^{220} + y_j^{002})$ or $(y_j^p + y_j^q) = (y_j^{002} + y_j^{220})$ or $(y_j^p + y_j^q)$

18

$= (y_j^{200} + y_j^{022})$ or $(y_j^p + y_j^q) = (y_j^{022} + y_j^{200})$

then we have $I_m \cdot v_{P_k} \cdot f(y_i) \cdot (y_j^p + y_j^q)$

$= I_m \cdot v_{P_k} \cdot f(y_i) \cdot y_j^{222} = I_m \cdot v_{P_k} \cdot$

$f(y_i)$ and therefore Y_k can be determined from the inputs and the variables y_i, $i \neq j$. If y_j^p, $y_j^q \in \left\{ y_j^{002}, y_j^{020}, y_j^{200} \right\}$, then there exists a third state, say s_r, such that $s_r = s_q(\tau_{y_i})$ for all $i \neq j$ and $N(s_r, I_m)$ $= N(s_q, I_m)(\tau_{y_k})$ for all I_m. If y_j^r represents the assignment in variable y_j for the state s_r, then next state function Y_k contains a combined product term $I_m \cdot v_{P_k}$

$\cdot f(y_i) \cdot (y_j^p + y_j^q + y_j^r)$ which is $I_m \cdot v_{P_k}$

$\cdot f(y_i) \cdot y_j^{222}$ which can be further reduced to $I_m \cdot v_{P_k} \cdot f(y_i)$. Therefore Y_k can be determined from the inputs and the variables y_i, $i \neq j$. If such a state s_r does not exist, then we can always specify a don't care to satisfy all the conditions of s_r. And again, Y_k can be determined from the inputs and the variables y_i, $i \neq j$.

The proof for the ternary case has been presented in detail, and a similar approach is applicable to an arbitrary R-valued case for $R \geq 4$. Q.E.D.

Appendix B: Proof for Theorem 3.2

The proof of this theorem is essentially identical to the proof in Theorem 3.1. There are, however, two points that need to be made. First, if $P(\prod_{i \neq j} \tau_{y_i}, \tau_{y_k})$ and $P(\prod_{i \neq 1} \tau_{y_i}, \tau_{y_k})$, then, based on Theorem 3.1, Y_k is independent of y_j and y_1. Second, if $S_i \cup S_m = S$ and $S_i \cap S_m = 0$, then the expressions $P(\prod_{y_i \in S_i} \tau_{y_i}, \tau_{y_k})$ and $P(\prod_{y_i \in S_m} \tau_{y_i}, \tau_{y_k})$ are identical. Q.E.D.

Appendix C: Proof for Theorem 3.5

Necessity: The necessity of condition (a) follows from the fact that if it is not satisfied, then there exist at least two states $s_a = s_b(\tau_{y_i})$ for all $y_i \in \sigma$ and an input vector I_k such that $N(s_a, I_k) \neq N(s_b,$

$I_k)(\tau_{y_i})$. Therefore, Y_j cannot be determined only from the input and $y_i \in \sigma$. Let $(s_a s_b, s_c s_d)$ be a dichotomy which is relevant to τ_{y_j}. If condition (b) is not satisfied, there is no $y_i \in \sigma$ which remains fixed at different values during the two distinct transitions, $s_a \to s_b$ and $s_c \to s_d$. It is therefore possible that the variables in σ may assume the same value y_σ during both of the transitions $s_a \to s_b$ and $s_c \to s_d$. Let S_σ be one of the internal states with the value y_σ for the variables in σ. In order for the circuit to realize the given flow table, $N(S_\sigma, I_k) = s_b(\tau_{y_j})$ and $N(S_\sigma, I_k) = s_d(\tau_{y_j})$ simultaneously. But $s_b \neq s_d$ (τ_{y_j}). Hence, $y_j \neq f_j(\sigma, I_k)$.

Sufficiency: Consider the transition from states s_a to state s_b under input I_k. Let states s_a and s_b have values y_σ^a and y_σ^b, respectively, for the variables in the set σ. Since condition (a) is satisfied, states in the same block of $\prod_{y_i \in \sigma} \tau_{y_i}$ with state s_a all have their next states in the same block of τ_{y_j}, that is, all their next states are assigned the same value for y_j. Furthermore, since condition (b) is satisfied, during the transition from s_a to s_b, the y-variables in σ will not assume any value y_σ which might be assumed during another transition, say from state s_c to state s_d, under input I_k where state s_d is assigned a y_j value different from state s_b. Therefore, all the states whose y variables in σ are assigned a value that might be assumed by during the transition from s_a to s_b have their next states in the same block of τ_{y_j} as state s_b. Hence all the vectors that may be assumed the y-variables in during the transition from y^a to y^b will yield a unique value of y_j. This is true for all transitions under any input vector I_k. Hence $y_j = f_j(\sigma, I_k)$. Q.E.D.

REFERENCES

[1] J.T. Butler and A.S. Wojcik, Guest editors' comments, IEEE transactions on Computers, Vol. C-30, No. 9, 1981.

[2] A.S. Wojcik, Relationships between Post and Boolean algebras with application to multi-valued switching theory, Coordinated Science Laboratory Report R-512, University of Illinois, Urbana, 1971.

[3] S.J. Sheafor, The design of multiple-valued asynchronous sequential circuits Ph.D. Thesis, University of Illinois, Urbana, Illinois, 1974.

[4] A.S. Wojcik, On the design of three-valued asynchronous modules, Proceedings of the Seventh International Symposium on Multiple-Valued Logic, Charlotte, North Carolina, May,1977.

[5] M. Davio, and J. - P. Deschamps, "Synthesis of Discrete Functions Using I^2L Technology," IEEE Transactions on Computers, Vol. C-30, No. 9, 1981

[6] K.C. Smith, Circuits of multiple-valued logic -- a tutorial and application, Proceedings of the Sixth International Symposium on Multiple-Valued Logic, Logan, Utah, 1976.

[7] T. Dao, K. Russell, D. Preedy, and E.J. McClausky, Multilevel I^2L with threshold gates, paper presented at the International Solid-State Circuits Conference, Philadelphia, PA., 1977.

[8] T.Dao, The year of multi-valued circuit realizations, lecture presented at the Seventh International Symposium on Multiple-Valued Logic, Charlotte, North Carolina, 1977.

[9] J. Hartmanis, On the state assignment problem for sequential machines-I, IRE Transactions on Electronic Computers. Vol. EC-10, No. 2 (1961), 157-165.

[10] R.E. Stearns and J. Hartmanis, On the state assignment problem for sequential machines-II, IRE Transactions on Electronic.

[11] J. Hartmanis and R.E. Stearns, Pair algebra and its application on automata, Information and Control, Vol. 7, No. 4 (1964), 485-507.

[12] J. Hartmanis and R.E. Stearns, Algebraic Structure Theory of Sequential Machines, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1966.

[13] P. Weiner and E.J. Smith, Optimization of reduced dependencies for synchronous sequential machines, IEEE Transaction on Electronic Computers, Vol. C-16, No. 12 (1967), 835-847.

[14] C.J. Tan, P.R. Menon, and A.D. Friedman, Structural Simplification and decomposition of asynchronous sequential circuits, IEEE Transactions on Electronic Computers, Vol. C-18, No. 9 (1969), 803-838.

[15] A.D. Friedman and P.R. Memnon, Theory and Design of Switching Circuits Computer Science Press, Inc., Woodland Hills, California, 1975.

[16] S.H. Unger, Asynchronous Sequential Switching Circuits, John Wiley and Sons, Inc., New York, 1969.

[17] A.S. Wojcik, Multi-valued asynchronous sequential circuits, Proceedings of the 1974 International Symposium on Multiple-Valued Logic, Morgantown, West Virginia, 1974.

[18] T.C. Yang, The decomposition of multi-valued sequential machines, Ph.D. Thesis, Illinois Institute of Technology, Chicago, Illinois, 1977.

A PREPROCESSING PROCEDURE METHOD IN TERNARY CLAUSE SELECTION

Shigeru IMANISHI and Noriaki MURANAKA

Faculty of Engineering, Kansai University
Yamate-cho 3-3-35, Suita-shi, Osaka, 564 Japan

ABSTRACT

We present an extension of the preprocessing procedure method used in the binary clause selection to the ternary clause selection. This extension is called a ternary preprocessing procedure method. When we make use of the ternary preprocessing procedure in the ternary clause selection, we can reduce a tree size. That is useful in a ternary prime implicant generation. We discuss the experimental results comparing the ternary clause selection and the ternary clause selection using the ternary preprocessing procedure.

I. INTRODUCTION

Slagle et al.[2] discuss a new algolithm for generating the prime implicants using semantic tree. In Slagle's method, a given logic function is expressed by a product-of-sum form. Literals in each logic sum term form a clause and a set of clauses corresponds to a given logic function. Slagle's method is extended to a clause selection by Kambayashi et al.[3], which is very useful in the prime implicant generation of binary logic function. A preprocessing procedure which is used in the clause selection is efficient because reductions of clauses in the set and literals included in clauses lead to reducing a tree size in the clause selection. The clause selection is extended to a ternary prime implicant generation. This extension is called a ternary clause selection[4,5].

Therefore in this paper, we present an extension of the preprocessing procedure to the ternary clause selection. The extention is called a ternary preprocessing procedure method (TPPM).

A ternary logic function is expressed in a product-of-sum form using coincidence functions [1]. Literals contained in each logic sum term form a clause. A set consists of clauses. When we make use of the TPPM in this set, we can reduce a tree size, which consists of nodes, branchs, remaining clauses, success nodes, and failure nodes. This reduction leads to reducing operation procedures in a process of ternary prime implicant generation. We discuss the experimental results comparing the ternary clause selection and the ternary clause selection using the TPPM.

II. A TERNARY PREPROCESSING PROCEDURE METHOD

II.1 A PRODUCT-OF-SUM FORM

A ternary variable takes on the truth value 0, 1 or 2. A product-of-sum form for a ternary n-variable logic function $F(x1,x2,...,xj,...,xn)$ (F, in short) using coincidence functions of Table 1 is as follows:[1]

$$F = (F (0,0,...,0) \lor Io (x1) \lor Io (x2)$$
$$...\lor Io (xn))$$
$$\cdot (F (1,0,...,0) \lor I1 (x1) \lor Io (x2)$$
$$...\lor Io (xn))$$
$$\cdot (F (2,0,...,0) \lor I2 (x1) \lor Io (x2)$$
$$...\lor Io (xn))$$
$$\cdot \cdot \cdot$$
$$\cdot (F (2,2,...,2) \lor I2 (x1) \lor I2 (x2)$$
$$...\lor I2 (xn)) \qquad (1)$$

21

where V and • are logic sum and logic product, respectively. F=1, Io(xj), Il(xj), and I2(xj) are called literals.

Table 1 The coincidence functions Ik (xj)

xj	Io (xj) 002 =xj	Il (xj) 202 =xj	I2 (xj) 220 =xj
0	0	2	2
1	2	0	2
2	2	2	0

II.2 A TERNARY PREPROCESSING PROCEDURE

A given ternary logic function is expressed by a product-of-sum form of Eq.(1) using literals. Literals contained in each logic sum term form a clause. If there exists a clause having literal 1, the clause is rewritten in other form, which is expressed by assistant-coincidence functions (1VIk(xj)) called literals of Table 2 as shown the following.

$$(1 \vee Ip (xu) \vee Iq (xv) \vee Ir (xw) \vee \dots)$$
$$= ((1VIp (xu)) \vee (1VIq (xv))$$
$$\vee (1VIr (xw)) \vee \dots) ;$$
$$p,q,r = 0,1,2 \qquad (2)$$

Table 2 The assistant-coincidence functions (1VIk (xj))

xj	(1VIo (xj)) 122 =xj	(1VIl (xj)) 212 =xj	(1VI2 (xj)) 221 =xj
0	1	2	2
1	2	1	2
2	2	2	1

Therefore, literals which constitute clause's elements are as follows.

If F (k1,k2,...,kn) =0 ,
         Ik1 (x1) ,Ik2 (x2) ,....,Ikn (xn)
If F (k1,k2,...,kn) =1 ,
         (1VIk1 (x1)) , (1VIk2 (x2)) ,...,
         (1VIkn (xn))
If F (k1,k2,...,kn) =2 ,
         we don't make a clause.

$$(3)$$

The clauses are elements of a set. Consider the following identities in the set.

$$(Ip (xj) \vee A) (Iq (xj) \vee A) (Ir (xj) \vee A) = A \quad (4)$$
$$(Ip (xj) \vee A) (Iq (xj) \vee A) ((1VIr (xj)) \vee B)$$
$$= 1 \cdot Ip (xj) Iq (xj) \vee A$$
$$= Ip (xj) Iq (xj) \vee A$$
$$= (Ip (xj) \vee A) (Iq (xj) \vee A) \cdot B \quad (5)$$
$$(Ip (xj) \vee A) ((1VIq (xj)) \vee B)$$
$$\cdot ((1VIr (xj)) \vee B)$$
$$= 1 \cdot Ip (xj) \vee A = (Ip (xj) \vee A) \cdot B \quad (6)$$
$$((1VIp (xj)) \vee B) ((1VIq (xj)) \vee B)$$
$$\cdot ((1VIr (xj)) \vee B) = B \qquad (7)$$
Where
$$A = Ik (x1) \vee Ik (x2) \vee \dots \vee Ik (xs) \dots ;$$
$$B = (1VIk (x1)) \vee (1VIk (x2)) \vee \dots$$
$$\vee (1VIk (xs)) \vee \dots ;$$
$$Ip (xj) Iq (xj) Ir (xj) = 0$$
$$p,q,r,k = 0,1,2 \qquad ; p \neq q \neq r \neq p ;$$
$$j,s = 1,2,\dots \qquad ; j \neq s$$

We will make use of Eq.(4)-Eq.(7) in an algolithm of the TPPM.

III. AN ALGOLITHM OF THE TERNARY PREPROCESSING PROCEDURE

III.1 AN ALGOLITHM

In order to reduce a tree size in ternary clause selection, an algolithm used in TPPM is presented. In the algolithm, a ternary logic function is given by a truth table. Let us explain the algolithm beginning at Procedure 1.

Procedure 1 We examine in the truth table whether a TPPM is available or not.

Procedure 1-(1) When we trace on a group of three function values corresponding

to $Io(xj)$, $Il(xj)$ and $I2(xj)$ for a logic variable xj in the truth table, if a sum of the number of function value 0 and 1 in the group is three, we choose one out of the following combinations.

function values

```
0  0  0 ------- three 0's
0  0  1 ------┐
0  1  0 ------├─ two 0's and one 1
1  0  0 ------┘
1  1  0 ------┐
1  0  1 ------├─ two 1's and one 0
0  1  1 ------┘
1  1  1 ------- three 1's
```

In the above figure, 0 or 1 is function value of the group given by the truth table.

Procedure 1-(2) If the sum in the group in Procedure 1-(1) isn't three, we can neglect the group because the group h.s one or more of the function value 2 and we can't use a preprocessing. And, we must trace on a next group.

Procedure 2 For the combination of three function values in the group obtained by Procedure 1-(1), we perform the following procedures.

Procedure 2-(1) If all of the function values in the group are 0, we can delete any two clauses out of the three clauses and the literal $Ik(xj)$ for the variable xj in the remaining clause due to Eq.4.

Procedure 2-(2) If all of the function values in the group are 1, we can delete any two clauses out of the three clauses and the literal $(1VIk(xj))$ for the variable xj in the remaining clause due to Eq.7.

Procedure 2-(3) If the function values in the group are 0 or 1, we have the following procedures.

Procedure 2-(3a) If there are one 0 and two 1's, the 0's clause is left as it is and, $(1VIq(xj))$ and $(1VIr(xj))$ in the

1's clauses are deleted due to Eq.6.

Procedure 2-(3b) If there are two 0's and one 1, the 0's clauses are left as they are and $(1VIr(xj))$ in the 1's clause is deleted due to Eq.5.

Procedure 3 Procedure 1 and Procedure 2 are repeated in all of the groups in the truth table.

Procedure 4 When Procedure 1-Procedure 3 have finished, we obtain a semantic tree's root (a node), which consists of the all of the remaining clauses. Here, the algolithm has fini_ned.

Procedure 5 Next, the ternary clause selection is applied to the node.

III.2 AN EXAMPLE OF THE TERNARY PREPROCESSING PROCEDURE

Let us consider a two variable function (x,y) given in Table 3. We obtain a node which is used as a semantic tree's root as follows.

Table 3 A two variable function $f(x,y)$

y	0 0 0	1 1 1	2 2 2
x	0 1 2	0 1 2	0 1 2
f	0 0 0	0 1 1	0 2 2

The following set of clauses in the ternary clause selection is directly obtained from Table 3.

$$\left\{ \begin{array}{l} [Io(x)\ VIo(y)] \\ [Il(x)\ VIo(y)] \\ [I2(x)\ VIo(y)] \\ [Io(x)\ VII(y)] \\ ((1VII(x))\ V\ (1V\ Il(y)) \\ ((1VI2(x))\ V\ (1V\ Il(y)) \\ [Io(x)\ VI2(y)] \end{array} \right.$$

Let us use the TPPM to this node. Then, when Procedure 2-(1) is performed for the variable x, we obtain the following clause $[Io(y)]$.

23

$$
\left.\begin{array}{l}
[\text{Io} (x) \text{ VIo } (y)] \\
[\text{I1} (x) \text{ VIo } (y)] \\
[\text{I2} (x) \text{ VIo } (y)]
\end{array}\right\} - [\text{Io} (y)]
$$

Again, when Procedure 2-(3b) is performed for the variable x, we obtain the following clauses $[\text{Io}(x)\text{VI1}(y)]$ and $[(\text{1VI1}(y))]$.

$$
\left.\begin{array}{l}
[\text{Io} (x) \text{ VI1 } (y)] \\
[(\text{1VI1} (x)) \text{ V } (\text{1V I1 } (y))] \\
[(\text{1VI2} (x)) \text{ V } (\text{1V I1 } (y))]
\end{array}\right\} -
$$
$$
- [\text{Io} (x) \text{ VI1 } (y)] , [(\text{1V I1 } (y))]
$$

Furthermore, when the Procedure 2-(1) is performed for the variable y, we obtain the following clause $[\text{Io}(x)]$.

$$
\left.\begin{array}{l}
[\text{Io} (x) \text{ VIo } (y)] \\
[\text{Io} (x) \text{ VI1 } (y)] \\
[\text{Io} (x) \text{ VI2 } (y)]
\end{array}\right\} - [\text{Io} (x)]
$$

Therefore, the node as the semantic tree's root is as follows.

$$
\left.\begin{array}{l}
[\text{Io} (y)] \\
[\text{Io} (x)] \\
[(\text{1VI1} (y))]
\end{array}\right\}
$$

Then, we apply the ternary clause selection method [4,5] of Procedure 5 to this node as showen in Fig.1.

From Fig.1, the following implicant is obtained. This is the prime implicant.

$$
x^{022} y^{022} y^{212} = x^{022} y^{012}
$$

IV. DISCUSSIONS

We have discussed the TPPM used in the ternary clause selection for the two or three variable functions given in the truth table of Table 4. The algorithm has been programmed on a FACOM-160F system using FORTRAN. The experiments of the two or three variable functions are shown in Table 5 or Table 6.

Fig.1-A ternary clause selection with the ternary preprocessing procedure method

From these tables, it is shown that the TPPMs are efficient for the functions of F1, F2, F4, F5, G1, G2, G3, G5, G6 and G8 because the number of the branchs, the remaining clauses, the failure nodes, and the non-prime implicants in the functions are reduced. In addition, we conjecture that the each sum function has the longest computation time of all of the two or three variable functions.

V. CONCLUSION

We have presented the ternary preprocessing procedure method used in the ternary clause selection. The ternary preprocessing procedure can be useful in the ternary prime implicant generation.

Table 4 Truth tables of the ternary two or three variable functions

F1	F2	F3	F4	F5	F6	F7	F8	F9
x	y	Sum	Carry	And	Or	0,1	1,2	2,0
0	0	0	0	0	0	0	1	2
0	1	1	0	0	1	1	2	0
0	2	2	0	0	2	0	1	2
1	0	1	0	0	1	1	2	0
1	1	2	0	1	1	0	1	2
1	2	0	1	1	2	1	2	0
2	0	0	0	0	2	0	1	2
2	1	0	1	1	2	1	2	0
2	2	1	1	2	2	0	1	2

G1	x	0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
G2	y	0 0 0 1 1 1 2 2 2 0 0 0 1 1 1 2 2 2 0 0 0 1 1 1 2 2 2
G3	z	0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
G4	Sum	0 1 2 1 2 0 2 0 1 1 2 0 2 0 1 0 1 2 2 0 1 0 1 2 1 2 0
G5	Carry	0 0 0 0 0 1 0 1 1 0 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 2
G6	And	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 1 0 1 2
G7	Or	0 1 2 1 1 2 2 2 2 1 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
G8	0 , 1	0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0
G9	1 , 2	1 1 2 1 2 1 2 1 1 1 2 1 2 1 1 1 1 2 2 1 1 1 1 2 1 2 1
G10	2 , 0	0 0 2 0 2 0 2 0 0 0 2 0 2 0 0 0 0 2 2 0 0 0 0 2 0 2 0

Table 5 Experiments' 1

functions		F1	F2	F3	F4	F5	F6	F7	F8	F9
branchs	(A)	9	9	18	11	10	10	15	12	8
		2	2	18	7	4	10	13	12	8
remaining		12	12	22	28	19	6	33	11	9
clauses	(B)	1	1	22	11	6	6	29	11	9
failure nodes		1	1	0	1	2	0	1	0	0
	(C)	0	0	0	0	0	0	0	0	0
implicants (D)		4	4	6	2	3	6	3	6	3
		1	1	6	2	1	6	3	6	3
non-prime		3	3	0	0	2	2	1	4	1
implicants (E)		0	0	0	0	0	2	1	4	1
prime implicants		1	1	6	2	1	4	2	2	2
	(F)	1	1	6	2	1	4	2	2	2
total computation		44	43	80	61	51	49	73	53	37
time (ms) (G)		19	19	80	40	26	52	68	55	38

upper : ternary clause selection
lower : ternary clause selection using ternary preprocessing procedure

Table 6　Experiments' 2

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
(A)	32 2	32 2	32 2	93 93	80 33	35 6	36 36	53 35	80 80	44 44
(B)	90 1	90 1	90 1	198 198	199 73	163 15	27 27	242 111	139 139	139 139
(C)	3 0	3 0	3 0	0 0	1 0	9 0	0 0	9 1	0 0	0 0
(D)	15 1	15 1	15 1	27 27	34 7	10 1	24 24	7 7	45 45	9 9
(E)	14 0	14 0	14 0	0 0	27 0	9 0	15 15	0 0	36 36	0 0
(F)	1	1	1	27	7	1	9	7	9	9
(G)	323 35	307 35	313 37	862 866	817 237	485 69	251 254	700 343	663 636	412 425

upper : ternary clause selection
lower : ternary clause selection using ternary preprocessing procedure

REFERENCES

[1] Porat,D.I.:"Three-valued digital systems",Proc. IEE.,Vol.116,No.6,pp.947-954 (June 1969).

[2] Slagle,J.R.,Chang,C.L. and Lee,R.C.:"A new algolithm for generating prime implicants",IEEE Trans.Comput.,C-19,pp.304-310 (Feb. 1970).

[3] Kambayashi,Y.,Okada,K. and Yajima,S.:"Prime implicant generation of logic functions using clause selection method",Trans.IECE Japan,J62-D,2,pp.89-96 (Feb. 1979).

[4] Imanishi,S. and Muranaka,N.:"Applications of clause selection method to ternary logical functions",Research Institute of Mathematical Seience,Kyoto University,Kookyuroku「Multiple-valued logic and its applications」,No.455, pp.94-107 (March 1982).

[5] Imanishi,S. and Muranaka,N.:"Applications of clause selection method to ternary prime implicant generation",Trans.IECE Japan,J65-D,7,pp.89-96 (July 1982).

A STRUCTURED DESIGN OF MULTIPLE-VALUED LSI/VLSI WITH BUILT-IN TESTING CAPABILITY

Samuel C. Lee and Krayim Santrakul

School of Electrical Engineering and Computer Science
University of Oklahoma
Norman, Oklahoma 73019

ABSTRACT

In this paper a structured design of subsystems of an MV LSI/VLSI chip with built-in testing capability is presented. This structured design which uses multiplexers and DFF's is obtained through the use of a tree-structured ASM chart. Since these circuits are highly structured and the procedure for obtaining the circuit values of the design from the ASM chart is rather straightforward and systematic, this design process may be adapted for automation.

A Multiple-Valued Built-In Block Observation (MVBILBO) is proposed. This circuit replaces the DFF's of a subsystem and provides the subsystem with the necessary hardware for having three operation modes: (1) basic system operation mode which provides the normal function of the DFF's, (2) Level Sensitive Scan Design (LSSD) operation mode which allows us to augment testing by controlling inputs and internal states and easily examining internal state behavior, and (3) Signature Analysis (SA) operation mode which can provide pseudorandom inputs to a subsystem to obtain a signature analysis test of the subsystem. By using these three operation modes of MVBILBO, two test generation/verification procedures are presented. One is for checking subsystems and the other for checking the complete circuit.

Both the design and testing methods are completely general and applicable to any MV LSI/VLSI chip design and testing.

1. Introduction

Recently, binary integrated circuit technology has moved from LSI to VLSI. This increase in gate count has brought a decrease in gate cost along with improvements in performance. All these attributes of VLSI are welcome by the industry. However, the problem never adequately solved by LSI is still with us and getting much worse: the problem of determining in a cost effective way, whether a component, module or board has been manufactured correctly [1-4]. It is known [5] that there are two major facets of the testing problem: test generation and test verification. Test generation is the process of enumerating stimuli for a circuit which will demonstrate its correct operation. Test verification is the process of proving that a set of tests are effective toward this end. It was also pointed out that due to their size and complexity, a complete exhaustive deterministic verification of an LSI, even for the binary logic, is almost impossible [5].

The design for testability techniques are divided into two categories [6]. The first category is that of the ad hoc technique for solving the testing problem. These techniques solve a problem only for a given design and are not generally applicable to all designs. This is contrasted with the second category of structured approaches. These techniques are generally applicable and usually involve a set of design rules by which designs are implemented. The objective of a structured approach is to reduce the sequential complexity of a network to aid test generation and test verification.

A considerable number of papers on the design for testability for LSI/VLSI have been published in the literature [7,8,9]. Among them the Level Sensitive Scan Design (LSSD) approach [7], Signature Analysis (SA) [8] and the Built-In Logic Block Observation (BILBO) [9] have received most attention. The LSSD technique [7] enhances both controlability and observability, allowing us to augment testing by controlling inputs and internal states and easily examining internal state behavior. The SA technique [8] is heavily reliant on planning done in the design stage and the key is to design a network which can stimulate itself. The BILBO approach [9] has ability to separate the network into combinational and sequential parts, and has the attribute of SA that is employing linear feedback shift registers.

Recently, a growing interest in the design of the multiple-valued digital system leads us to believe that sooner or later this technology will also move to LSI/VLSI. The objective of this paper is to find a MV LSI/VLSI design technique which has the following features:

(1) A simple and systematic design procedure, adaptable for automation.

(2) A module-based circuit structure so that the parallel (simultaneous) testing of all modules at each level may be applied.

(3) A Built-in logic block for circuit testing included in each subsystem.

27

(4) Both the exhaustive (deterministic) and random (sample) testings offered by the built-in testing logic hardware so that the format may be used for checking relatively small logic blocks and the latter for checking larger logic blocks.

(5) A hierarchical testing procedure that can not only detect the faults in the circuit, if any, but also locate them within modules.

A summary of the paper by section is as follows. Section 2 presents a structured MV LSI/VLSI subsystem design using ASM chart. The built-in testing logic circuit is described in section 3. The test generation/verification of MV LSI/VLSI is presented in Section 4.

2. Structured Design of MV LSI/VLSI

In this section a structured design of MV LSI/VLSI using ASM chart [10,11,12,13] is presented. Since one of our objectives is to find a simple and systematic scheme for designing highly structured MV LSI/VLSI, no function minimization in this discussion will be considered. This method starts with describing the design using a tree-structured ASM chart as shown in Fig. 1, which is to be realized by a highly structured multiplexers/DFF's circuit of Fig. 2. From these two diagrams we see that the input values of the multiplexers of the circuit in Fig. 2 can be found directly from the ASM chart of Fig. 1 and thus this design process can be automated by a computer. This process is best illustrated by an example. Without loss of generality, consider the ASM chart of Fig. 3 where the 3-valued logic is used. The realization of this chart is shown in Fig. 4. The input values to the multiplexers of Fig. 4 are obtained from the state variables, and transition paths from one state to another indicated on the ASM chart. For example, the values of 1 and 2 at the inputs of the first multiplexers of the two blocks are obtained from the state variable values $q_1 = 1$ and $q_2 = 2$ of the final state ⑥ of the transition path from state ① to state ⑥ as indicated in dark line. The rest of input values to the multiplexers are found in a similar manner. Both multiplexers and DFF's of this design can be realized by I^2L [14]

3. Design of Multiple-Valued Built-In Logic Block Observation (MVBILBO) circuit

Before presenting MVBILBO, several MV logic gates and flip-flops which will be used in the construction of MVBILBO, are first introduced

AND operation:	$x + y = max(x,y)$
OR operation:	$x \cdot y = min(x,y)$
NOT operation	$\bar{x} = p - x$ where $p = m-1$
NOR operation:	$\overline{x + y} = p - max(x,y)$
NAND operation:	$\overline{x \cdot y} = p - min(x,y)$
EXCLUSIVE-OR operation:	$x \oplus y = |x-y|$

In this paper the same circuit symbols used in the binary gate are adopted for the MV circuit and the MV flip-flops to be used in the MVBILBO are D-flip-flops [12] . Fig. 5(a) shows an MVBILBO circuit. Z_i are the outputs from the combinational multiplexer circuit, Q_i are the state variables, C_1 and C_2 are control lines, S_{IN} and S_{OUT} are data scan in and data scan out, respectively. This circuit has three modes of operation controlled by the two control lines C_1 and C_2.

(1) Basic System operation mode (C_1C_2=pp). When C_1C_2=pp, the MVBILBO of Fig. 5(a) will be reduced to the circuit shown in Fig 5(b). Under this operation the Z_i values are loaded into D_i, and the outputs are available on Q_i for system operation. This would be normal register function.

(2) LSSD operation mode (C_1C_2=00). When C_1C_2= 00, the MVBILBO register takes on the form of a linear shift register as shown in Fig. 5(c). Data scan-in input to the left, through some NOT gates, and basically lining up the registers into a single scan path, until the data scan-out is reached.

(3) Signature analysis operation mode (C_1C_2=p0).

When C_1C_2=p0 in this mode, MVBILBO register takes on the attributes of a linear feedback shift register of a maximal length with multiple linear inputs. If inputs to the MVBILBO registers, Z_i can be controlled to fixed values. Under this mode of operation, the MVBILBO will output a sequence of patterns which are very close to random patterns. They will be used in the signature analysis of MV LSI/VLSI circuit which is discussed in the next section.

A subsystem may be divided into two parts: multiplexer (combinational) circuit and DFF (memory) circuit (see Fig. 6(a)). In order for subsystem to be testable, the DFF circuit part is replaced by MVBILBO as shown in Fig. 6(b). This provides the subsystem with built-in testing capability.

4. Test Generation/Verification of MV LSI/VLSI

In this section, two efficient test generation/verification procedures are presented. One is the LSSD which is to be applied to relatively small circuits whose inputs and outputs are accessible to the test engineer and the other is the signature analysis which is mainly for testing large subsystems and LSI/VLSI circuits.

The testing of an MV LSI/VLSI involves the following two major parts:

A. Test of subsystems

This test is conducted at the end of the fabrication of the subsystems and before they are

interconnected together. Either a LSSD or a signature analysis test will be used depending on the size of the subsystem being tested.

a. The LSSD test

This is an exhaustive and deterministic test of the subsystem. The test procedure is as follows:

Step 1: Apply a set of input values I_i to the inputs of the subsystem (we assume that both its inputs and outputs are accessible to the test engineer)

Step 2: Set C_1C_2=00. Scan in an initial internal state $(q_1...,q_n)$ through the data scan-in input terminal (S_{IN}).

Step 3: Set C_1C_2=pp. Record the outputs and compare them with prestored expected output values.

Step 4: Keep the input values I_i fixed and set C_1C_2=00. Scan in another internal state $(q_1...,q_n)$ through the S_{IN} and at the same time record the outputs $(Q_1...,Q_n)$ from the S_{OUT} compare this set of values with prestored expected next-state values.

Step 5: Repeat steps 2-4 until all the states are exhausted.

Step 6: Repeat steps 1-5 for all possible values of inputs.

b. The signature analysis test

In this case where the size of the subsystem is too large for an LSSD test; a signature analysis test is then recommended. Fig. 8 represents a subsystem where I_i are the system inputs which are accessible from chip pins J_j are internal inputs from other subsystems. This test method consists of the following steps:

Step 1: Set C_1C_2=00. Scan in the initial internal state $(q_1...,q_n)$ through the data scan-in input terminal.

Step 2: Apply a set of values to the chip pins I_i.

Step 3. By setting C_1C_2=p0, where p=m-1, the MVBILBO will generate a set of pseudorandom numbers which are fed to the J_j inputs internally as shown in Fig. 7.

Step 4: Set C_1C_2=pp, namely making MVBILBO operate in its basic system operation mode, to produce the next state values (Q_k).

Step 5: Repeat steps 3 and 4 N times to ensure that the subsystem is thorough checked under the fixed system inputs I_i and random input J_j.

Step 6: Set C_1C_2=00 and shift in a new set of internal state variables $(q_1...,q_n)$. At the same record the shift-out data set (Q_k) from the scan-out (S_{OUT}) and compare it as a signature with a set of prestored expected next state values. If they are identical, it means that the subsystem has passed this portion of the test and is ready to receive the next test; otherwise a fault of this portion of the circuit has been detected.

Step 7: Apply a new set of I_i and go to steps 3, 4, and then 5.

Step 8: Repeat step 6 and 7 until the subsystem is checked for a sufficient large number of different sets of I_i and q_k.

Note that to shorten testing time, parallel testings of these subsystems are recommended whenever possible.

B. Test of the MV LSI/VLSI Circuit

After all the subsystems are tested and proven to be faultless, they will then be interconnected together. All the MVBILBO's of the subsystems will be connected serially, that is, a scan-out terminal will be connected to the scan-in terminal of one of its adjacent subsystems and the first (last) scan-in (scan-out) terminal is connected to the scan-in (scan-out) pin of the chip.

Since the number of tests needed for a complete exhaustive test of the MV LSI/VLSI chip using the LSSD method is astronomical, a random signature analysis test is used instead. The way to conduct such a test is similar to the one described above, except there will be no internal inputs (J_j) and all the input values for the I_i will be the random numbers generated by the MVBILBO (see Fig. 8). Let N_i be the number of tests applied to the subsystem i for each set of fixed values of I_i. The recommended number of tests applied to the entire circuit for a set of fixed valued of I_i is $N_1 \times N_2 ... \times N_s$ or larger, where s denotes the number of subsystems of the MV LSI/VLSI circuit.

5. Conclusion

A completely general synthesis procedure for realizing any multiple-valued sequential logic using a structured MV circuit with multiplexers/DFF's and built-in testing logic hardware has been presented. Two efficient test generation/verification procedures, the LSSD and the signature analysis, are recommended. The LSSD is recommended for checking relatively small circuitry (subsystem) and signature analysis for checking LSI/VLSI circuit. Both the synthesis and testing procedures are suitable for machine automation.

29

REFERENCES

1. M.A. Breuer, ED., Diagnosis and Reliable Design of Digital Systems. Woodland Hills, CA: Computer Science Press, 1976.

2. S. Bisset, "Exhaustive Testing of Microprocessors and Related Devices: A practical Solution," in Dig. 1977 Semiconductor Test Symp., Oct., 1977, pp. 38-41.

3. R. A. Frohwerk, "Signature Analysis: A new digital fields service method," Hewlett-Packard J., May, 1977, pp. 2-8.

4. C.W. Weller, "An Engineering Approach to IC Test System Maintenance," in Dig. 1977 Semiconductor Test Symp., Oct. 1977, pp. 144-145.

5. T.W. Williams and K.P. Parker, "Design for Testability: A Survey," IEEE Trans. on Comp., Vol. C-31, No. 1 Jan., 1982, pp. 2-15.

6. T.W. Williams and K.P. Parker, "Testing Logic Networks and Design for Testability," Computer, Oct., 1979, pp. 9-21.

7. E.B. Eichelberger and T.W. Williams, " A Logic Design Structure for LSI Testability," J. Design Automation Fault-Tolerant Comput., Vol. 2, May 1978, pp. 165-178.

8. H.J. Nadig, "Signature Analysis: Concepts, examples, and guide lines, "Hewlett-Packard J. May 1977, pp. 15-21.

9. B. Koenemann, J. Mucha, and G. Zwiehoff, "Built-In Logic Block Observation Techniques," in Dig. 1979 Test conf., Oct. 1979, pp. 37-41.

10. C.R. Clare, Designing Logic Systems Using State Machines, McGraw-Hill, New York, 1973.

11. S.C. Lee, Digital Circuits and Logic Design, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1976.

12. D. Winkel and F. Prosser, The Art of Digital Design: An introduction to Top-Down Design, Prentice Hall 1980.

13. W.S. Wojciechowski, "Structured Digital Systems Design in Multiple-Valued Logic," Proc. the Twelfth Intn. Symp. on Multiple-Valued Logic, Paris, France, 1982.

14. J. H. Pugsley and C.B. Silio, Jr., "Some I^2L circutis for Multiple-Valued Logic," Proc. the Eight Intn. Symp. on Multiple-Valued Logic, Rosemont, Ill., 1978, pp. 23-31.

15. M.S. Wills, "A Behavioral Model and Triggering Modes for MVL R-flops," Proc. the Eight Intn. Symp. on Multiple-Valued Logic, Rosemont, Ill., 1978, pp. 226-234.
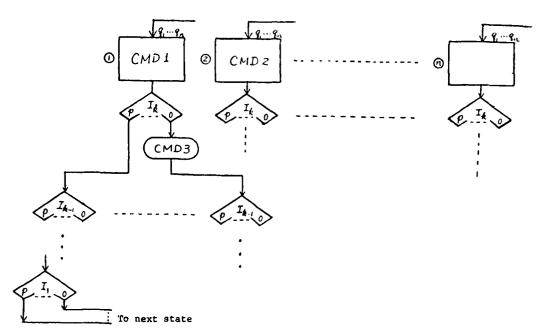
Fig.1 A tree structured ASM Chart
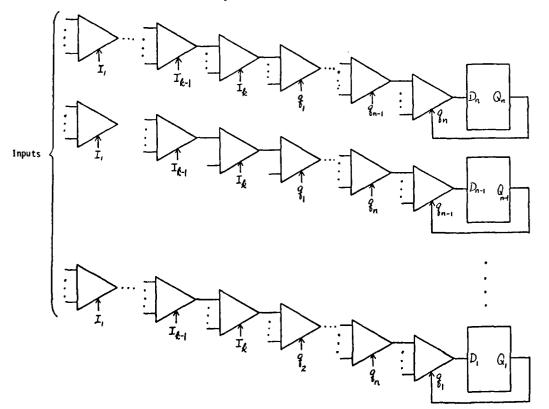
Fig. 2 A highly structured multiplexers/DFF's circuit realization.
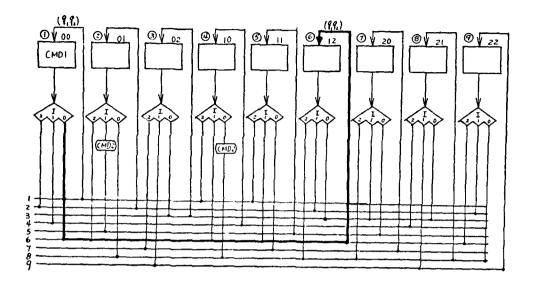
Fig. 3 An example of 3-valued ASM chart
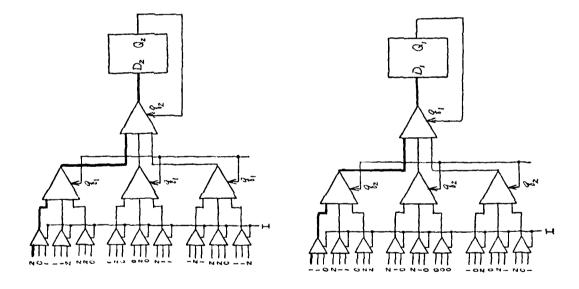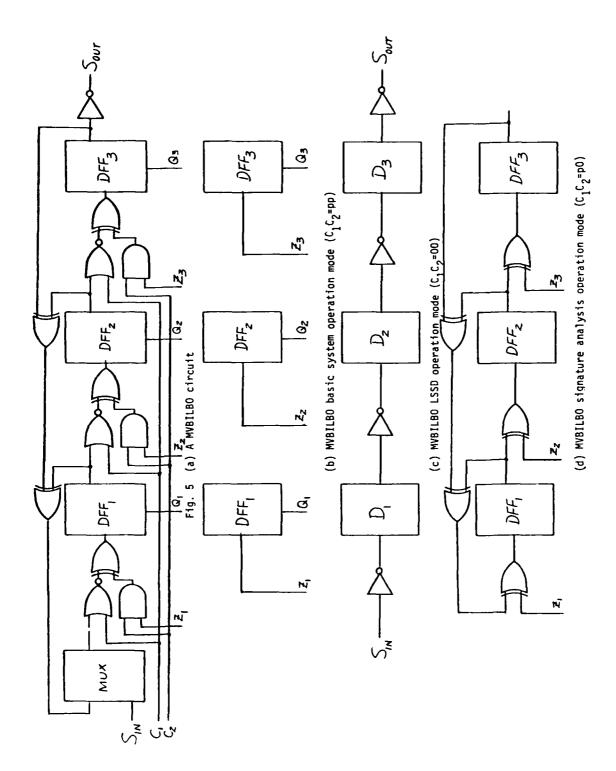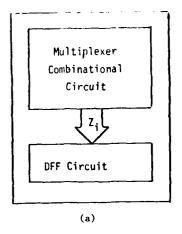
Fig. 4 The multiplexers/DFF's circuit realization
of the ASM chart of Fig. 3.

32

Fig. 5 (a) A MVBILBO circuit

(b) MVBILBO basic system operation mode $(C_1C_2=pp)$

(c) MVBILBO LSSD operation mode $(C_1C_2=00)$

(d) MVBILBO signature analysis operation mode $(C_1C_2=p0)$

33
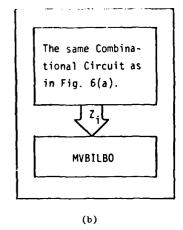
Fig. 6 (a) A subsystem
(b) A subsystem with its DFF circuit
replaced by an MVBILBO
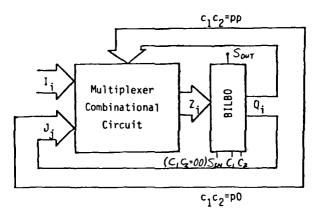
Fig. 7 Test generation/verification for a subsystem
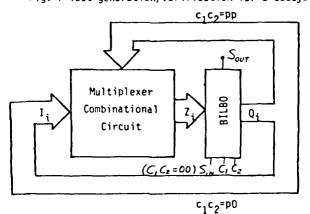
Fig. 8 Test generation/verification for a
complete MV LSI/VLSI circuit.

34

Session 1B
Philosophy I

AXIOMATIC CHARACTERIZATION AND COMPARATIVE ANALYSIS

OF

PREFERENCE ON DESIRABILITY AND POSSIBILITY

by Osamu KATAI and Sousuke IWAI

Dept. of Precision Mechanics, Faculty of Engineering

Kyoto University, Sakyo-ku, Kyoto 606, JAPAN

ABSTRACT

A complete axiomatic system which characterizes the preference relations (the relation of comparative desirability or comparative possibility in different cases) under Maximin (Minimax), Maximax), Bayesian, Probabilistic, Plausibilistic and Fuzzy theoretic evaluations is introduced. A set of theses on preference relations is presented, and their validity is tested under the different systems. Based on these results, the commonalities and peculiarities of the preference relations are elucidated and a systematic way of identifying the underlying evaluation method of arbitrarily given preference data is constructed.

1. INTRODUCTION

We will characterize various evaluation methods for desirability or possibility of cases under uncertainty.

We will proceed by introducing certain axiomatic systems which are characteristic of these methods. More precisely, we will clarify the general laws of comparison between different cases (i.e., the laws of comparative desirability or possibility) which are summarized by axiomatic systems known as "preference logic"[1,2].

We will consider here *Maximin (Minimax), Maximax* and *Bayesian* evaluations of desirability, *Probabilistic* and *Plausibilistic* evaluations of possibility, as well as *Fuzzy* theoretic evaluation for which both interpretations are possible. Maximax evaluation refers to the extreme case (the most optimistic case) of Hurwiczian evaluation and also is regarded as an evaluation of possibility[3]. Plausibility was first introduced by Rescher[4] and provides a systematic and rigorous evaluation of truthfulness (credibility) of cases (represented as propositions), by which reasoning under hypotheses becomes possible.

2. PREFERENCE LOGICS - AXIOMATIC SYSTEMS CLARIFYING COMPARATIVE DESIRABILITY OR POSSIBILITY

In the following, we will adopt the following principle that every case will be specified as a proposition's being true. Let us compare the desirability or possibility of a case where the proposition, say p, is true with that of another case where the proposition, say q, is true. This line of approach for the comparative analysis of desirability was first adopted by Jeffrey[1] and Rescher[2].

Maximin Preference Logic

Comparison of desirability by Maximin method is based on the evaluation of each case as the minimum value of states within the case. Suppose, for instance, that we have to take into account three fundamental propositions, say p, q and r, for specifying states, and we have eight states from $w_1 = p \wedge q \wedge r$ (the state in which p, q and r are true) to $w_8 = \sim p \wedge \sim q \wedge \sim r$ (neither of them being true). We will call each of these states a "possible world". Let $V(w_i)(i=1, 2, \ldots, 8)$ be the evaluation of the possible world w_i. Then $V(p)$, the evaluation of the case of p's being true, is given as Min $\{ V(p \wedge q \wedge r), V(p \wedge \sim q \wedge r), V(p \wedge q \wedge \sim r), V(p \wedge \sim q \wedge \sim r) \}$.

If $V(p) \geq V(q)$, i.e., the case of p's being true is "more preferable" (more desirable) or "equivalent" to the case of q's being true, then we have a preference relation "R" between propositions p and q written as

$$p \, R \, q.$$

Also, strict preference relation "P" and indifference relation "I" are defined as

$$p \, P \, q \equiv p \, R \, q \wedge \sim (q \, R \, p)$$
$$p \, I \, q \equiv p \, R \, q \wedge q \, R \, p.$$

In the following, we will clarify the general laws to which relation "R" is subject. First, it should be noted that we have

$$\text{if} \vdash^* A \to B, \text{ then } A \, R \, B$$

for arbitrary propositions A and B, where $\vdash^* C$ means that proposition C is valid in the propositional calculus (PC). Also, it is clear that under arbitrary truth value assignments

$$V(w_i) = 0 \text{ (false) or } 1 \text{ (true) for all } w_i.$$

A R B means that $A \to B$ is true. Hence, it is to be expected that binary relations between propositions have properties similar to those of implicational relation "$\to$" as represented by theorems of propositional calculus as follows: $\vdash^* (p \to q) \vee (q \to p)$(connectedness), $\vdash^* (p \to q) \wedge (q \to r) \to (p \to r)$(transitivity), $\vdash^* (p \to q) \wedge (p \to r) \to (p \to q \vee r)$(disjunctive continuity), $\vdash^* (p \to q) \wedge (p \wedge r \to q \wedge r)$(conjunctive continuity), $\vdash^* (p \to q) \to (\sim q \to \sim p)$ (contraposability), etc.[5]

In the sequel, we will focus our attention on the following general laws (theses) regarding R, some of which were introduced by von Wright, Chisholm & Sosa et al[2] in their logico-philosophical analysis of preference. We will clarify which of them are valid under various evaluations of desirability or possibility as mentioned in the introduction. From these analyses we can construct an axiomatic system for the preference relations under each evaluation method, by which means the characterization and comparison of preference relations may be elucidated.

List of Theses

1. $p R q \vee q R P$ *(connectedness)*

2. $p R q \wedge q R r \rightarrow p R r$ *(transitivity)*

3. $(p \vee q) R p$ *(positive disjunctive monotonity)*

4. $p R (p \vee q)$ *(negative disjunctive monotonity)*

5. $p R q \rightarrow (p \vee r) R (q \vee r)$ *(disjunctive continuity)*

5'. 5 holds provided that

$\vdash^* r \equiv p$ or $\vdash^* r \equiv q$, and $\vdash^* \sim (p \wedge q)$

$\left(\begin{array}{c} \text{or equivalently,} \\ p R (\sim p \wedge q) \rightarrow p R (p \vee q) \quad \text{and} \\ (\sim p \wedge q) R p \rightarrow (p \vee q) R p \end{array} \right)$

5''. 5 holds provided that

$\vdash^* \sim (r \wedge p)$ and $\vdash^* \sim (r \wedge q)$

$\left(\begin{array}{c} \text{or equivalently} \\ (p \wedge \sim r) R (q \wedge \sim r) \rightarrow (p \vee r) R (q \vee r) \end{array} \right)$

6. $p R q \rightarrow (p \wedge r) R (q \wedge r)$ *(conjunctive continuity)*

6'. 6 holds provided that

$\vdash^* (p \wedge \sim r) \equiv (q \wedge \sim r)$

$\left(\begin{array}{c} \text{or equivalently} \\ (p \vee r) R (q \vee r) \rightarrow (p \wedge \sim r) R (q \wedge \sim r) \end{array} \right)$

7. $p R (p \vee q) \rightarrow p R (\sim p \wedge q)$ *(right subtractability)*

8. $(p \vee q) R p \rightarrow (\sim p \wedge q) R p$ *(left subtractability)*

9. $(\sim p) R (\sim q) \rightarrow q R p$ *(contraposability)*

10. $p I (p \vee p)$ *(idempotency)*

11. $(p \vee q) I (q \vee p)$ *(disjunctive commutativity)*

12. $(\sim \sim p) I p$ *(double negation)*

13. $(\sim p) R p \wedge q R (\sim q) \rightarrow q R p$ *(order linearity)*

14. $p R q \rightarrow (p \wedge r) R (q \wedge r) \vee (p \wedge \sim r) R (q \wedge \sim r)$ *(weak conjunctive continuity)*

15. $(p \vee q) R p \vee (p \vee q) R q$ *(weak positive disjunctive monotonity)*

16. $(q \wedge r) R p \rightarrow q R p \vee r R p$ *(left extensibility)*

17. $q R p \vee r R p \rightarrow (q \vee r) R p$ *(left mergeability)*

18. $p R q \wedge p R r \rightarrow p R (q \vee r)$ *(right mergeability)*

19. $p R q \rightarrow (p \wedge \sim q) R (\sim p \wedge q)$ *(continuity on symmetric difference)*

20. $(q \vee r) R p \rightarrow q R p \vee r R p$ *(left separability)*

21. $(p \vee \sim p) I (q \vee \sim q)$ *(indifference of tautologous cases)*

It is not difficult to show that valid theses under Maximin evaluation are 1, 2, 4, 5, 5', 5'', 8, 10, 11, 12, 13, 14, 15, 18,

20 and 21. For instance, thesis 5 is easily verified by noting that

$$V(p \vee r) \triangleq \min \left\{ V(w_i) \mid p \vee r \text{ is true in } w_i \right\}$$
$$= \min \left\{ V(p), V(r) \right\}.$$

Counter example for thesis 6 is given as

$V(w_i) = 2$ for any w_i where both q and r are true.
$V(w_i) = 1$ for the other possible world w_i's.

It is clear in this case that $V(p) = V(q) = V(p \wedge r) = 1$ while $V(q \wedge r) = 2$, which contradicts thesis 6.

Next, we search for the minimal set of theses from which all the valid theses are derivable. Of course, we presume the following rules of inference together with the axiomatic system of PC (propositional calculus).

rule 1 (rule of substitution)
If $\vdash A$, then we have $\vdash A'$, where A' is obtained from A by replacing some occurrences of a propositional variable with a well-formed formula (proposition) in PC.

rule 2 (rule of tautologous indifference)
If $\vdash^* A \equiv B$, then we have $\vdash A I B$.

It is clear that thesis 5' and 5'' follow from 5; 8 from 4, 10 12 from from PC; 13 from 2 and 5; 14 from 1, 2, 4 and 5; 15 from 1 and 5; 18 from 2 and 5; 20 from 2 and 4; and 21 from rule 2. For instance, thesis 13 is verified as

$$\vdash q R (\sim q) \overset{5}{\rightarrow} (q \vee q) R (\sim q \vee q) \overset{10}{\rightarrow} q R (\sim q \vee q), \text{ and}$$
$$\vdash (\sim p) R p \overset{5}{\rightarrow} (\sim p \vee p) R (p \vee p) \overset{10}{\rightarrow} (\sim p \vee p) R p, \text{ and hence}$$
$$\vdash q R (\sim q) \wedge (\sim p) R p \overset{\text{rule 2}}{\rightarrow} q R (\sim q \vee q) \wedge (\sim q \vee q) R p \overset{2}{\rightarrow} q R p.$$

Moreover, it is not difficult to show that theses 1, 2, 4 and 5 and rules 1 and 2 together with the axiomatic system of PC constitute a complete axiomatic system for the preference logic under Maximin evaluation of desirability. Namely, it is possible to derive every valid thesis (laws regarding preference relations) from this system.

[Axiomatic System of Maximin Preference Logic]

Axioms:
1. $\vdash p R q \vee q R p$ (connectedness)
2. $\vdash p R q \wedge q R r \rightarrow p R r$ (transitivity)
4. $\vdash p R (p \vee q)$ (negative disjunctive monotonity)
5. $\vdash p R q \rightarrow (p \vee r) R (q \vee r)$ (disjunctive continuity)

Rules:
rule 1 (rule of substitution)
rule 2 (rule of tautologous indifference)
(together with the axiomatic system of PC).

Maximax Preference Logic

We proceed to Maximax preference logic which is based on the evaluation of desirability of a case by the maximum value of states within the case. Namely,

$$V(A) = \max_{w_i \subset W(A)} V(w_i),$$

where

$$W(A) = \left\{ w_i \mid A \text{ is true in } w_i \right\}.$$

37

Let $V'(w_i)$ denote $-V(w_i)$ for all w_i's. Then it is obvious that $V'(A)$(in Maximin evaluation) is equal to $-V(A)$(in Maximax evaluation) for an arbitrary proposition A. Hence A R' B in Maximin preference based on V' is equivalent to B R A in Maximax preference based on V. Thus we obtain a complete axiomatic system of Maximax preference logic (together with the axiomatic system of PC) as follows:

[Axiomatic System of Maximax Preference Logic]

Axioms:
1. ⊢ p R q ∨ q R p (connectedness)
2. ⊢ p R q ∧ q R r → p R r (transitivity)
3. ⊢ (p ∨ q) R p (positive disjunctive transitivity)
5. ⊢ p R q → (p ∨ r) R (q ∨ r) (disjunctive continuity)

Rules:
rule 1 (rule of substitution)
rule 2 (rule of tautologous indifference)

Bayesian Preference Logic

The preference logic under Bayesian evaluation of desirability was first examined by Saito[6] who succeeded the researches of Jeffrey and Rescher[1,2]. Saito's conjectural axiomatic system is composed of axioms 1, 2, 5', 7 and 8 and rules 1 and 2 together with PC. Based on Jeffrey's characterization theorem[7] on Bayesian evaluation, we can verify that Saito's system is complete provided that the number of fundamental propositions is finite (i.e., when we have only a finite number of possible worlds) and with no possible world having a zero probability. Hence we have

[Axiomatic System of Bayesian Preference Logic]

Axioms:
1. ⊢ p R q ∨ q R p (connectedness)
2. ⊢ p R q ∧ q R r → p R r (transitivity)
5'. ⊢ p R (~p ∧ q) → p R (p ∨ q)
(disjunctive continuity)
⊢ (~p ∧ q) R p → (p ∨ q) R p
7. ⊢ p R (p ∨ q) → p R (~p ∧ q)
(right subtractability)
8. ⊢ (p ∨ q) R p → (~p ∧ q) R p
(left subtractability)

Rules:
rule 1 (rule of substitution)
rule 2 (rule of tautologous indifference)

Next, we proceed to the preference logic on possibility evaluation which clarifies the more possible (probable or plausible) case between two alternatives.

Probabilistic Preference Logic

First, we examine the preference logic under probabilistic evaluation of possibility, i.e.,

$$V(A) = \text{Prob.} (A) = \sum_{w_i \in W(A)} V(w_i),$$

where $V(w_i)$ is the probability of world w_i's being true. According to the results of Fishburn[8], we can show that the following system (together with PC) constitutes a complete axiomatic system for the finitary case (i.e., the case where we have only a finite number of possible worlds).

[Axiomatic System of Probabilistic Preference Logic]

Axioms:
1. ⊢ p R q ∨ q R p (connectedness)
2. ⊢ p R q ∧ q R r → p R r (transitivity)
3. ⊢ (p ∨ q) R p (positive disjunctive monotonity)
5''. ⊢ (p ∧ ~r) R (q ∧ ~r)
→ (p ∨ r) R (q ∨ r) (disjunctive continuity)
6'. ⊢ (p ∨ r) R (q ∨ r) → (p ∧ ~r) R (q ∧ ~r)
(conjunctive continuity)

Rules:
rule 1 (rule of substitution)
rule 2 (rule of tautologous indifference)

Plausibilistic Preference Logic

"Plausibility" evaluates the possibilities of cases (propositions) being true through the notion of "modal category", that is, an increasing sequence $M_0, M_1, \ldots, M_i, \ldots, M_n$ of sets of propositions (in PC) which satisfy the following conditions[4]:

i) $A \in M_0$ iff ⊢* A,
ii) if $A \in M_i$, then $A \in M_j$ for $j \geq i$,
iii) M_i: a set of mutually consistent propositions,
iv) for any i, if $A \in M_i$ and ⊢* A → B, then $B \in M_i$,
v) for any A, there exists i such that $A \in M_i$,
vi) for any i, if A, A' $\in M_i$, then $A \wedge A' \in M_i$.

The propositions in M_0 (i.e., PC tautologies) have the highest plausibility, and those in $M_1 - M_0$ have the second highest and so on. Hence the preference logic in this case is given as

$$A \, R \, B \text{ iff } \min \left\{ i \mid A \in M_i \right\} \geq \min \left\{ j \mid B \in M_j \right\}.$$

By introducing an increasing sequence of sets of possible world given as

$$W_i \triangleq \bigcap_{A \in M_i} W(A) \qquad i = 0, 1, 2, \ldots, n,$$

we can show that

$A \in M_i$ iff $W(A) \supseteq W_i$, for any proposition (of PC) A and for any i.

Moreover, we can verify that the converse also holds. Namely, if we have an increasing sequence $W_0, W_1, W_2, \ldots$ of sets of possible worlds, the sequence $M_0, M_1, M_2, \ldots$ given above, i.e.,

$$M_i \triangleq \left\{ A \mid W(A) \supseteq W_i \right\},$$

satisfies condition i)–vi). Attributing value $V(w_k) = i$ to the possible world w_k's in $W_i - W_{i-1}$ (where W_{-1} is set as ϕ), we can show that

$$A \, R \, B \text{ iff } V(\sim A) \leq V (\sim B)$$

for arbitrary propositions A and B, where V is assumed to be evaluated by Maximax method, i.e.,

$$V(A) = \max_{w_i \in W(A)} V(w_i).$$

Hence we arrive at the relation that

$$A \, R \, B \text{ iff } (\sim B) R' (\sim A),$$

where R' is a Maximax preference relation. It follows from the axiomatic system of Maximax preference logic that the following system constitutes a complete axiomatic system

for Plausibility preference (of course, we must presume the axiomatic system of PC).

[Axiomatic System of Plausibilistic Preference Logic]

Axioms:
1. ⊢ p R q ∨ q R p (connectedness)
2. ⊢ p R q ∧ q R r → p R r (transitivity)
3. ⊢ (p ∨ q) R p (positive disjunctive monotonity)
6. ⊢ p R q → (p ∧ r) R (q ∧ r) (conjunctive continuity)

Rules: rule 1 (rule of substitution)
rule 2 (rule of tautologous indifference)

Fuzzy Preference Logic

We have clarified various preference relations based on Maximin, Maximax, Bayesian, Probabilistic and Plausibilistic evaluation of possible world. Fuzzy theoretic evaluation of propositions, on the other hand, does not refer to the possible worlds, but applies directly to the evaluation (desirability or possibility) of propositions themselves[9, 10].

In the following, we will confine ourselves to the evaluation of propositions without implications (for, in Fuzzy logic, A → B is not equivalent to ~A ∨ B, and involves another aspect of preference logics which is beyond the scope of the present paper). In this case, the evaluation laws for logical connectives are given as

$$V(A \wedge B) = \min \{V(A), V(B)\},$$
$$V(A \vee B) = \max \{V(A), V(B)\}, \text{ and}$$
$$V(\sim A) = 1 - V(A).$$

It can be readily seen that all the theses in the list except 4, 8 and 21 are valid, and that rule 2 is inapplicable here. Also by referring to the axiomatic system of PC, it can be verified that the following system (together with PC) constitute a complete axiomatic system of Fuzzy theoretic preference.

[Axiomatic System of Fuzzy Preference Logic]

Axioms:
1. ⊢ p R q ∨ q R p (connectedness)
2. ⊢ p R q ∧ q R p → p R r (transitivity)

3. ⊢ (p ∨ q) R p (positive disjunctive monotonity)
5. ⊢ p R q → (p ∨ r) R (q ∨ r) (disjunctive continuity)
9. ⊢ (~ p) R (~ q) → q R p (contraposability)
10. ⊢ p I (p ∨ p) (idempotency)
11. ⊢ (p ∨ q) I (q ∨ p) (disjunctive commutativity)
12. ⊢ (~ ~ p) R p (double negation)

Rule: rule 1 (rule of substitution)

3. COMPARATIVE ANALYSES OF THE PREFERENCE LOGICS

By examining the validity of each thesis in the list by the above mentioned axiomatic systems or by constructing counter examples of the thesis, as shown in the discussion of Maximin preference logic, we can obtain the validity results summarized in Table 1, where the double circle ◎ represents the valid theses adopted as axioms and the open circle ○ represents the valid theses (theorems) in each of the preference logics.

First of all, it should be noted that theses 1 and 2 are the common axioms and show that each preference relation is a total order.

As mentioned in the introduction, Bayesian and Maximin evaluation can be regarded as an evaluation of desirability in the strict sense (i.e., they cannot be regarded as evaluations of possibility). Also, Probability and Plausibility are notions concerned only with possibility. Maximax and Fuzzy theoretic evaluations, however, are concerned with both sides. Hence, thesis 8 (left subtractability) characterizes the desirability preferences in the strict sense, whereas thesis 5' (disjunctive continuity) characterizes them in the broad sense. Also, theses 3 (positive disjunctive monotonity), 16 (left extensibility) and 17 (left mergeability) characterize the possibilistic preferences in the broad sense. On the other hand, there is no thesis characterizing the latter in the strict sense (although 6', conjunctive continuity, and 19, continuity on symmetric difference, characterize Probabilistic, Plausibilistic and Fuzzy theoretic preferences).

From the Table, the peculiarities and commonalities cha-

Theses/Rules	1	2	3	4	5	5'	5"	6	6'	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	r1	r2
Bayes	◎	◎				◎				◎	◎		○	○	○	○							○		◎	◎
Maximin	◎	◎		◎	◎	○	○					○	○	○	○	○	○	○			○		○	○	◎	◎
Maximax	◎	◎	◎		◎	○	○			○			○	○	○	○	○	○	○	○	○		○	○	◎	◎
Fuzzy	◎	◎	◎		◎	○	○	○	○	○		◎	◎	◎	◎	○	○	○	○	○	○	○	○		◎	
Probability	◎	◎	◎			◎			◎	○		○	○	○	○	○	○	○	○	○	○		○		◎	◎
Plausibility	◎	◎	◎					◎	○	○		○	○	○	○	○	○	○	○	○		○		○	◎	◎

Table 1. Validity of the theses and rules 1 and 2 of inference under each preference logic.

racteristic of each preference can be noted. As for the former, for instance, theses 4 (negative disjunctive monotonity) and 7 (right subtractability) characterize Maximin preference from the others; 14 (weak conjunctive continuity) and 15 (weak positive disjunctive monotonity) do Bayesian preference from the others; and Rule 2 (rule of tautologous indifference) and 21 (indifference of tautologous cases) are characteristic of Fuzzy preference.

As for the commonalities among the preferences, for instance, theses 5 (disjunctive continuity), 18 (right mergeability) and 20 (left separability) are commonly valid in Maximin, Maximax and Fuzzy preferences. Also, thesis 9 (contraposability) is common to both Fuzzy and Probability preferences.

Suppose that we are given a set of preference relations (data) such as $\{A_1RA_2, A_3PA_4, A_5IA_6, \ldots\}$ where A_i's are propositions (of PC) specifying cases, and we wish to search for compatible interpretations of the relations, i.e., we want to identify the basic method of evaluation underlying the data as

Maximin (pessimistic evaluation) method, Maximax (optimistic evaluation), Bayesian (evaluation by expected utility) and so on. (Here we do not presume that the preference is based on desirability or on possibility, although in practical situations this is usually known beforehand.)

Incompatible interpretations can be detected by discovering counter examples of the preference logics axioms. First, we have to clarify whether or not the data include counter examples of thesis 1 or 2, which are the common axioms to all preference logics. If there is one, we can say that the data are not based on any of the evaluation methods discussed here. If there is none, we then search for the counter examples of theses 3–12, which are the remaining theses adopted as preference logic axioms, in order to clarify further incompatibility (although in practical situations theses 10–12 sometimes denote trivial instances).

One systematic way of conducting this search is shown in Fig. 1, where "$\overset{yes}{\to}$" ("$\overset{no}{\to}$") represents that there is at
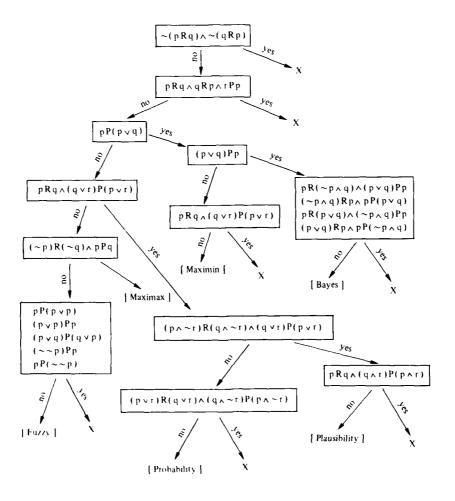
Fig. 1. A systematic search method for the compatible interpretation of arbitrarily given set of preference relations.

40

least one (no) counter example of the formula in the box, "X" represents that there is no possible interpretations, and [Maximin], for instance, means that it is possible to interpret the data as being based on Maximin evaluation method. Also, it should be noted that if thesis 1 hold, i.e., if there is no counter example of 1, then we have

$$\sim (p R q) \equiv q R p.$$

Hence, for instance, the negation of thesis 2 can be written as

$$p R q \wedge q R r \wedge r P p.$$

Although not indicated in the Figure, the validity of rule 2 should also be checked, except in the case of [Fuzzy].

4. CONCLUSION

We have clarified a complete axiomatic system describing the preference relation based on each evaluation method of desirability or possibility, and by which we can make analyses of comparative desirability or comparative possibility.

These qualitative analyses, though apprehending only certain aspects of the evaluation methods, enable us to obtain various information useful for the interpretation of data concerning these evaluation methods as discussed in section 3.

REFERENCES

1) Jeffrey, R. C.: *The Logic of Decision*, McGraw-Hill (1965).
2) Rescher, N.: Semantic foundations for the logic of preference, in N. Rescher (ed.): *The Logic of Decision and Action*, pp. 37–70, Univ. of Pittsburgh Press (1966).
3) Schlaifer, R.: *Analysis of Decision under Uncertainty*, McGraw-Hill (1969).
4) Rescher, N.: *Hypothetical Reasoning*, pp. 45–56, North-Holland (1964).
5) Hilbert, D. and Ackermann, W.: *Grundzüge der Theoretischen Logik*, Springer-Verlag (1949).
6) Saito, S.: Semantical considerations on the logic of preference, *Philosophy of Science*, Vol. 7, pp. 117–131 (1974) (in Japanese).
7) Jeffrey, R. C.: Axiomatizing the logic of decision, in C. A. Hooker et al. (eds.): *Foundation and Application of Decision Theory*, Vol. 1, pp. 227–231, Reidel (1978).
8) Fishburn, P. C.: *Utility Theory for Decision Making*, pp. 194–195, J. Wiley (1969).
9) Goguen, J. A.: The logic of inexact concepts, *Synthese*, Vol. 19, pp. 325–373 (1968–9).
10) Giles, R.: Łukasiewicz logic and fuzzy set theory, *International Journal of Man-Machine Studies*, Vol. 8, pp. 313–327 (1976).

QUOTIENT ALGEBRAS FOR LOGICS OF IMPRECISION*

Michael Katz

School of Education, Haifa University, Haifa, Israel

1. THREE LOGICAL SYSTEMS

In the 1981 ISMVL (Oklahoma City) I proposed two logical systems to deal with the problem of imprecision and measurement error in science, calling them "the logic of inexactness" and "the logic of approximation" (see [6]). In the 1982 ISMVL (Paris) I showed how to apply the logic of approximation to quantum theory, and how to reduce it, in certain cases, to a third, perhaps simpler, logical system (see [7]). In the 1983 ISMVL (Kyoto) I would like to consider the algebras of these three logical systems.

Let me start by introducing the systems in a manner somewhat different from that in the previous works mentioned above, a manner which, among other things, will enable me to avoid the notion of truth-values.

Our basic ingredients are a formal propositional language, L, and a set, X, of valuations of formulae of L. As is quite common (see, e.g., Dalla Chiara [1]), we think of the valuations as corresponding to "possible worlds" or to "states" of a mechanical system. But our valuations differ from ordinary ones in that they range in [0,1] rather than in {0,1}. The reason for this is that we perceive the statements of L as being imprecise (perhaps due to measurement errors), and a given valuation estimates the degree of imprecision (or error) of every statement in a given world (or state).

Thus, every valuation is a [0,1]-valued function on the set of formulae of L. Looking at the situation from the "complementary" point of view, every formula of L is a [0,1]-valued function on the set X of valuations. It is this second view that we shall adopt in this paper. In fact, in order to simplify our arguments, we shall assume that every [0,1]-valued function on X is a formula of L.

The language L contains the usual connectives of conjunction (∧), disjunction (∨), impli-

---

cation (→) and negation (¬). If ψ and θ are formulae of L we stipulate that for every $x \in X$

$$(\psi \wedge \theta)(x) = \max\big(\psi(x), \theta(x)\big)$$

$$(\psi \vee \theta)(x) = \min\big(\psi(x), \theta(x)\big)$$

$$(\psi \rightarrow \theta)(x) = \max\big(0, \theta(x) - \psi(x)\big)$$

$$(\neg\psi)(x) = 1 - \psi(x).$$

These semantic rules reverse the traditional ones of the Łukasiewicz logic. The reversal arises from the replacement of truth-values by degrees of error. Thus in our systems, no error (0) replaces absolute truth and maximal error (1) replaces absolute falsity; conjunctions maximize the error while disjunctions minimize it; the error in an implication statement is measured by the degree to which the error in the consequent exceeds that in the antecedent; and a negation statement is an implication statement in which the error in the consequent is maximal. Some of these ideas are due to Scott ([11], [12]) and are similar to those of Giles in [4] and subsequent papers (as well as in his Comment on [12]).

The rules given above are shared by the three logical systems discussed in this paper. The systems differ from each other in how they handle the notion of deduction.

By a deduction expression we mean an expression of the form $\psi \vdash \theta$ (read: "θ is deducible from ψ") where ψ and θ are formulae of L. (We shall write $\psi\vdash$ when every formula of L is deducible from ψ, and $\vdash\theta$ when θ is deducible from every formula of L). This notion is easily extendable to the case where we have (finite) sets of formulae of L on the left side, or on both sides, of the symbol $\vdash$ (see Scott [11], Katz [6], [7]), but this extension will not be needed for our purposes here.

In the logic of inexactness (LI) we say that $\psi \vdash \theta$ if the inexactness (i.e., error) in ψ always exceeds that in θ, that is, if for all $x \in X$

$$\psi(x) \geqslant \theta(x).$$

In the logic of approximation (LA) to be able to deduce θ from ψ is to be able to reduce the error in θ as much as we wish (thus, to approximate exactness as closely as we wish) by making the error in ψ small enough. So we say that $\psi \vdash \theta$

if for every ε we can find a δ s.t. for all $x \in X$

$$\psi(x) < \delta \Rightarrow \theta(x) < \varepsilon .$$

(Here, and in the sequel, ε and δ denote positive real numbers, while $\Rightarrow$ denotes implication outside L).

In the third logical system, which we shall denote by LL as deductions in it resemble those in the conventional Łukasiewicz Logic (see, e.g., Smiley's Comment on [12]), we say that $\psi \vdash \theta$ if θ is error-free wherever ψ is error-free, i.e., if for every $x \in X$

$$\psi(x) = 0 \Rightarrow \theta(x) = 0 .$$

If one of the conditions above is satisfied for the formulae ψ and θ we say that the deduction $\psi \vdash \theta$ is valid in the appropriate logic. It is easy to see that every deduction valid in LI is also valid in LA, and every deduction valid in LA is also valid in LL. Simply note that an equivalent definition of $\psi \vdash \theta$ in LI is provided by stipulating that for every ε and every x

$$\psi(x) < \varepsilon \Rightarrow \theta(x) < \varepsilon ,$$

and an equivalent definition of it in LA (see my proof in [7]) can be provided by requiring that for every sequence $\{x_n : 1 \leq n < \infty\}$ of elements of X

$$\lim_{n \to \infty} \psi(x_n) = 0 \Rightarrow \lim_{n \to \infty} \theta(x_n) = 0 .$$

Symbolically we express these facts by writing

$$LI \subset LA \subset LL.$$

To see that these are strict inclusions let $X = [1, \infty)$. Then, if $\psi(x) = 1$ for every $x \in X$, we have $\psi \vdash$ in each of the three systems; if $\psi(x) = 1/2$ for every $x \in X$, we have $\psi \vdash$ in LA and LL but not in LI; and if $\psi(x) = 1/x$ for every $x \in X$, we have $\psi \vdash$ only in LL. On the other hand, the three systems agree on $\vdash \theta$; in each of them this is valid if and only if θ is constantly zero over X.

For each of the three logics described in this section it is not hard to see that the relation $\equiv$ defined by

$$(\equiv) \qquad \psi \equiv \theta \text{ iff } \psi \vdash \theta \And \theta \vdash \psi$$

is an equivalence relation over the set of formulae of L. Thus, we can try to define appropriate operations on the collection of equivalence classes modulo $\equiv$ in order to obtain a quotient algebra (in which $\equiv$ reduces to $=$ and $\vdash$ to a partial order which we shall denote by $\leq$) and then to investigate the properties of this algebra.

As we shall see in the following section, this is easy to do in the case of LI. However, in the cases of LA and LL there are problems with the definitions of $\rightarrow$ and $\neg$. In Section 3, we shall give two alternative definitions for these operations in the case of LL and show that one of the

algebras obtained is Boolean and the other "nearly" Heyting. Then, in Section 4, we shall ask how these new operations behave in the case of LA. Concluding remarks will be made in Section 5.

2. THE ALGEBRA OF LI

In the case of LI ($\equiv$) becomes

$$\psi \equiv \theta \text{ iff } \forall x \in X (\psi(x) = \theta(x)).$$

Writing $\varphi \in L$ for "φ is a formula of the language L" and denoting by $[\varphi]$ the equivalence class of φ modulo $\equiv$, we can now define the following operations over the collection, LI/$\equiv$, of these classes:

$$[\psi] \wedge [\theta] = [\psi \wedge \theta] = \{\varphi \in L : \forall x \in X . \varphi(x) = \max(\psi(x), \theta(x))\}$$

$$[\psi] \vee [\theta] = [\psi \vee \theta] = \{\varphi \in L : \forall x \in X . \varphi(x) = \min(\psi(x), \theta(x))\}$$

$$[\psi] \rightarrow [\theta] = [\psi \rightarrow \theta] = \{\varphi \in L : \forall x \in X . \varphi(x) = \max(0, \theta(x) - \psi(x))\}$$

$$\neg[\psi] = [\neg \psi] = \{\varphi \in L : \forall x \in X . \varphi(x) = 1 - \psi(x)\}$$

This is the well-known method of obtaining algebras of fuzzy sets (see, e.g., De Luca and Termini [2]), and thus most parts of the following theorem are also well known.

Theorem 1. The algebra LI/$\equiv$, with the operations defined above, is a distributive lattice and for every $\psi, \theta, \chi \in L$ it satisfies

(1) $\quad \neg\neg[\psi] = [\psi]$

(2) $\quad \neg([\psi] \vee [\theta]) = \neg[\psi] \wedge \neg[\theta]$

(3) $\quad \neg([\psi] \wedge [\theta]) = \neg[\psi] \vee \neg[\theta]$

(4) $\quad ([\psi] \rightarrow [\theta]) \wedge [\theta] = [\theta]$

(5) $\quad ([\psi] \rightarrow [\psi]) \wedge [\theta] = [\theta]$

(6) $\quad ([\psi] \rightarrow [\theta]) \wedge ([\psi] \rightarrow [\chi]) = [\psi] \rightarrow ([\theta] \wedge [\chi])$

(7) $\quad \neg([\psi] \rightarrow [\psi]) \vee [\theta] = [\theta]$

(8) $\quad [\psi] \rightarrow \neg([\psi] \rightarrow [\psi]) = \neg[\psi] .$

The first three laws of this theorem (the "law of double negation" and the "De Morgan laws"), together with the fact that LI/$\equiv$ is a distributive lattice, mean that we have here a De Morgan algebra (see again De Luca and Termini [2]). The remaining five laws (the laws for implication, (4), (5), and (6), and the laws relating implication to negation, (7) and (8)) are all but one of those used in Rasiowa and Sikorski [10], pp. 123-124, to define the notion of a pseudo-Boolean algebra (more often called a Heyting algebra). The one missing law is

(9) $\quad [\psi] \wedge ([\psi] \rightarrow [\theta]) = [\psi] \wedge [\theta] .$

In LI we can only prove

$$\psi \wedge \theta \vdash \psi \wedge (\psi \rightarrow \theta) .$$

And LI/$\equiv$ deviates from a Heyting algebra also in that (i) it satisfies the law of double

43

negation and (ii) it does not satisfy the law of non-contradiction (see the following section).

To conclude this section we define two distinguished elements, t and f, of $LI/\equiv$ by

$$t = \{\varphi \in L : \vdash \varphi\} = \{\varphi \in L : \forall x \in X.\varphi(x) = 0\}$$

$$f = \{\varphi \in L : \varphi \vdash\} = \{\varphi \in L : \forall x \in X.\varsigma(x) = 1\} ,$$

and then we have

__Theorem 2:__ For every formula φ of L

(10) $\qquad f \leqslant [\varphi] \leqslant t$

(11) $\qquad \neg[\varphi] = [\varphi] \to f$

(12) $\qquad t \to [\varphi] = [\varphi] .$

We note that the second of these laws is the one used in the theories of Boolean algebras and pseudo-Boolean algebras to define negation from implication.

3. THE CASE OF LL

In the case of LL $(\equiv)$ becomes

$$\psi \equiv \theta \quad \text{iff} \quad \forall x \in X \big(\psi(x) = 0 \quad \text{iff} \quad \theta(x) = 0\big)$$

Then we can define the following operations on $LL/\equiv$.

$$[\psi] \wedge [\theta] = [\psi \wedge \theta] =$$

$$= \{\varphi \in L : \forall x \in X \big(\varphi(x) = 0 \text{ iff } \max\big(\psi(x),\theta(x)\big) = 0\big)\}$$

$$[\psi] \vee [\theta] = [\psi \vee \theta] =$$

$$= \{\varphi \in L : \forall x \in X \big(\varphi(x) = 0 \text{ iff } \min\big(\psi(x),\theta(x)\big) = 0\big)\} .$$

But we cannot define $\neg[\psi]$ and $[\psi] \to [\theta]$ by means of the classes $[\neg\psi]$ and $[\psi \to \theta]$ since such definitions will depend on the choice of the particular $\psi \in [\psi]$ and $\theta \in [\theta]$.

We shall now suggest two ways of defining the operations $\to$ and $\neg$ and $LL/\equiv$. The first one is the following.

$$\neg[\psi] = \{\varphi \in L : \forall x \in X \big(\varphi(x) = 0 \text{ iff } \psi(x) \neq 0\big)\}$$

$$[\psi] \to [\theta] = \neg[\psi] \vee [\theta] =$$

$$= \{\varphi \in L : \forall x \in X \big(\varphi(x) = 0 \text{ iff } \big(\psi(x) \neq 0 \text{ or } \theta(x) = 0\big)\big)\}$$

To these we add the following definitions of t and f

$$t = \{\varphi \in L : \vdash \varphi\} = \{\varphi \in L : \forall x \in X.\varphi(x) = 0\}$$

$$f = \{\varphi \in L : \varphi \vdash\} = \{\varphi \in L : \forall x \in X.\varphi(x) \neq 0\}$$

and then we have

__Theorem 3:__ The algebra $LL/\equiv$, with the operations defined above, is a distributive lattice and for every $\psi \in L$ it satisfies

(13) $\qquad [\psi] \wedge \neg[\psi] = f$

(14) $\qquad [\psi] \vee \neg[\psi] = t .$

These are the familiar "law of non-contradiction" and "law of excluded middle", and thus we have here a Boolean algebra (which justifies the definition of $\to$ by $\neg$ and $\vee$). Hence, as is well known, all the identities (1)-(12) of the preceding section hold in $LL/\equiv$.

The second way of defining negation in $LL/\equiv$ is by setting, for every $\psi \in L$ (with the t and f of this section),

$$[\psi] = \begin{cases} t & \text{if } \psi \vdash \text{ (i.e., if } \forall x \in X.\psi(x) \neq 0) \\ f & \text{if } \psi \not\vdash \text{ (i.e., if } \exists x \in X.\psi(x) = 0) \end{cases}$$

And two, almost identical, implications can be defined in connection to this negation, as follows,

$$[\psi] \underset{1}{\to} [\theta] = \begin{cases} t & \text{if } \psi \vdash \theta \text{ (i.e., if (a):} \\ & \forall x \in X.\psi(x) = 0 \to \theta(x) = 0) \\ [\theta] & \text{if } \psi \not\vdash \theta \text{ (i.e., if (b):} \\ & \exists x \in X.\psi(x) = 0 \text{ & } \theta(x) \neq 0) \end{cases}$$

$$[\psi] \underset{2}{\to} [\theta] = \begin{cases} t & \text{if } \psi \vdash \theta \text{ (i.e., if (a))} \\ [\psi \wedge \theta] & \text{if } \psi \not\vdash \theta \text{ (i.e., if (b)).} \end{cases}$$

__Theorem 4:__ With the new $\neg$ and $\underset{1}{\to}$, $LL/\equiv$ satisfies, for every $\psi,\theta \in L$, (4), (5), (7)-(13). With the new $\neg$ and $\underset{2}{\to}$, $LL/\equiv$ satisfies, for every $\psi,\theta,\chi \in L$, (5)-(13).

The conclusion from this theorem is that we have two algebras here, both very close to a *Heyting algebra. The one property missing for* $LL/\equiv$ with $\neg$ and $\underset{1}{\to}$ to qualify as a Heyting algebra is (6). *The one missing for* $LL/\equiv$ *with* $\neg$ and $\underset{2}{\to}$ is (4). *The negation used in both of them has the Heyting algebra properties of satisfying the law of non-contradiction but not the laws of double negation and excluded middle.*

4. THE CASE OF LA

In the logic of approximation, LA, $(\equiv)$ becomes

$$\psi \equiv \theta \quad \text{iff for every sequence } \{x_n : 1 \leqslant n < \infty\}$$

$$\lim_{n \to \infty} \psi(x_n) = 0 \iff \lim_{n \to \infty} \theta(x_n) = 0$$

In the sequel we denote sequences as above by X_n and write $\lim \varphi(X_n)$ for $\lim_{n \to \infty} \varphi(x_n)$.

The first two operations on $LA/\equiv$ are again easy to define. We set for every $\psi,\theta \in L$

$$[\psi] \wedge [\theta] = [\psi \wedge \theta] =$$

$$= \{\varphi \in L : \forall X_n \big(\lim \varphi(X_n) = 0 \iff \lim \max\big(\psi(X_n),\theta(X_n)\big) = 0\big)\}$$

$$[\psi] \vee [\theta] = [\psi \vee \theta] =$$

$$= \{\varphi \in L : \forall X_n \big(\lim \varphi(X_n) = 0 \iff \lim \min\big(\psi(X_n),\theta(X_n)\big) = 0\big)\}$$

Once again the operations $\neg$ and $\to$ cannot be defined simply by taking the classes $[\neg\psi]$ and $[\psi\to\theta]$. So let us look at the two proposals of the preceding section and see to what extent they can be applied to the present case.

We start with the first negation operator of the preceding section. When we deal with $LA/\equiv$, the definition becomes

$$\neg[\psi] = \left\{\varphi \in L: \forall X_n \left(\lim \varphi(X_n)=0 \text{ iff } \lim \psi(X_n)\neq 0\right)\right\} .$$

To this we attach the following definitions of $\to$ and f (while the definition of t remains as in the cases of $LI/\equiv$ and $LL/\equiv$):

$$[\psi] \to [\theta] = \left\{\varphi \in L: \forall X_n \left(\lim \varphi(X_n) = 0\right.\right.$$
$$\left.\left. \text{iff } \left(\lim \psi(X_n)\neq 0 \text{ or } \lim \theta(X_n)=0\right)\right)\right\}$$

$$f = \left\{\varphi \in L : \varphi \vdash\right\} =$$

$$= \left\{\varphi \in L : \forall X_n . \lim \varphi(X_n) \neq 0\right\} .$$

Now, the problem with these definitions is that the elements $\neg[\psi]$ and $[\psi] \to [\theta]$ do not always exist. For instance, if $X = [1,\infty)$ and $\psi(x) = 1/x$ for every $x \in X$ then $\neg[\psi]$ is undefined, and hence so is $[\psi] \to [\theta]$ for many θ's.

The following theorem, whose proof is quite easy, shows what can be recovered from Theorem 3 in the case of LA.

Theorem 5: The algebra $LA/\equiv$, with the operations defined above, is a distributive lattice. Any element $[\psi]$ of $LA/\equiv$ for which $\neg[\psi]$ exists is Boolean in the sense that (13) and (14) hold for it. The laws (1)-(12) hold in the algebra whenever all the elements involved are defined.

When we move to the second set of negation and implication operations of the preceding section we have, in the case of LA, the following definitions (with t and f of this section):

$$\neg[\psi] = \begin{cases} t & \text{if } \psi \vdash \text{ (i.e., if } \forall X_n . \lim \psi(X_n) \neq 0) \\ f & \text{if } \psi \not\vdash \text{ (i.e., if } \exists X_n . \lim \psi(X_n) = 0) \end{cases}$$

$$[\psi] \underset{1}{\to} [\theta] = \begin{cases} t & \text{if } \psi \vdash \theta \text{ (i.e., if (a)}^*: \\ & \forall X_n . \lim \psi(X_n) = 0 \to \lim \theta(X_n) = 0) \\ [\theta] & \text{if } \psi \not\vdash \theta \text{ (i.e., if (b)}^*: \\ & \exists X_n . \lim \psi(X_n) = 0 \ \& \ \lim \theta(X_n) \neq 0) \end{cases}$$

$$[\psi] \underset{2}{\to} [\theta] = \begin{cases} t & \text{if } \psi \vdash \theta \text{ (i.e., if (a)}^*) \\ [\psi \wedge \theta] & \text{if } \psi \not\vdash \theta \text{ (i.e., if (b)}^*) \end{cases}$$

The last theorem of this section tells us that with the new operations we have now two "near" Heyting algebras, precisely as in the case of $LL/\equiv$.

Theorem 6: The algebra $LA/\equiv$ with the new $\neg$ and $\underset{1}{\to}$ ($\neg$ and $\underset{2}{\to}$) satisfies the same laws which are satisfied by $LL/\equiv$ with the $\neg$ and $\underset{1}{\to}$ ($\neg$ and $\underset{2}{\to}$, respectively) of Theorem 4 above.

5. CONCLUDING REMARKS

The results obtained in this paper, while technically not very difficult to prove, are significant from a philosophical point of view. They help us understand the meaning of truth, falsity, negation and implication in our logic of approximation and draw comparisons with other logics.

In this paper, as noted in its second paragraph, we tried to avoid the controversy concerning multiple truth-values, speaking instead about degrees of imprecision. We did define truth and falsity (t and f) terms for each of our algebras, but in the cases of $LL/\equiv$ and $LA/\equiv$ these definitions provide another, more subtle, way around the controversy. In the case of $LL/\equiv$ a formula of L is in f if it has no zero points and in the case of $LA/\equiv$ if it never approaches zero. In both cases (and in the case of $LI/\equiv$) a formula of L is in t only if it is constantly 0 ("fully true").

Thus, although there is a multitude of degrees of imprecision in LL and LA, we are in line with those (like Miller [8]) who claim that any two statements which are not fully true are equally false. Moreover, a statement which is nowhere fully true is everywhere fully false, regardless of whether the error involved is 0.1 or 1.0. And two formulae coincide, deductively, if they have zeros in the same places (or in the case of LA, approach zero along the same sequences), regardless of what happens in other points (or along other sequences).

In view of these results it seems only natural that our first negation and implication operations, in the cases of $LL/\equiv$ and $LA/\equiv$, turn out to be Boolean. We would like to add here that it should also come as no surprise that our second definition of negation and implication, in these two cases, lead to structures very close to a Heyting algebra. We find already in Gödel [5] and Dummett [3] that operations like, e.g., our $\underset{1}{\to}$ are studied in connection to systems close to intuitionistic logic.

Finally we note that if we take $LA/\equiv$ (or $LL/\equiv$) with the two sets of negation and implication operations, we obtain an algebra resembling in some aspects the quasi-pseudo-Boolean algebras used in Rasiowa [9] to characterize "constructive logic with strong negation" and in other aspects the "pluralistic logic" proposed in Dalla Chiara [1] as a formal framework for quantum theory. This ties well with our perception of the logic of approximation as a potential logic for science.

REFERENCES

[1] Dalla Chiara, M.L., Logical foundations of quantum mechanics, in E. Agazzi (ed.), Modern Logic - A Survey, D. Reidel Publ. Co., Dordrecht, Holland, 1980, 331-351.

[2] De Luca, A. and Termini, S., Algebraic properties of fuzzy sets, J. Math. Anal. Appl. 40, 1972, 372-386.

[3] Dummett, M., A propositional calculus with denumerable matrix, J. Symbolic Logic, 24, 1959, 97-106.

[4] Giles, R., A non-classical logic for physics, *Studia Logica*, 33, 1974, 397-415.

[5] Gödel, K., Zum intuitionistischen Aussagen-kalkül, *Ergebn. Math. Kolloq.* Heft 4, 1933,40.

[6] Katz, M., Two systems of multi-valued logic for science, *Proceedings of the 11th International Symposium on Multiple-Valued Logic*, IEEE Computer Society Press, New York, 1981, 175-182.

[7] Katz, M., The logic of approximation in quantum theory, *J. Phil. Logic*, 11, 1982, 215-228.

[8] Miller, D., The accuracy of predictions, *Synthese*, 30,1975, 159-191.

[9] Rasiowa, H., *An Algebraic Approach to Non-Classical Logics*, North Holland, Amsterdam, 1974.

[10] Rasiowa, H., and Sikorski, R., *The Mathematics of Metamathematics*, PWN, Warsaw, 1963.

[11] Scott, D., Completeness and axiomatizability in many-valued logic, in L. Henkin et al. (eds.), Proceedings of the Tarski Symposium *(Proceedings of Symposia in Pure Mathematics 25)*, Amer. Math. Soc., Providence, R.I., 188-197.

[12] Scott, D., Does many-valued logic have any use? in S. Körner (ed.), *Philosophy of Logic*, Basil Blackwell, Oxford, 1976, 64-95 (including Comments by T.J. Smiley, J.P. Cleave and R. Giles).

IN THE LABYRINTH OF MANY VALUED LOGICS

"où l'Indécis au Précis se joint"

PAUL VERLAINE

Salvatore GUCCIONE

Istituto di Fisica Teorica
Università di Napoli
N A P O L I , Italy

Settimo TERMINI

Istituto di Cibernetica
del C.N.R.
ARCO FELICE (Napoli),Italy

Roberto TORTORA

Istituto di Matematica
Università di Napoli
N A P O L I , Italy

Introduction

The present paper[1] reviews *some* aspects of recent developments of *Many-Valued Logics*. It then does not aim at any completeness, rather the analysis presented here endeavours to provide only a possible Ariadne's thread to help the reader to find some paths that seemed to the authors worth following in the *many-valued* Labyrinth of the various types of problems of the subject. Also the bibliography, then, aims at no completeness. The classical book by N. Rescher (1966) will be taken as a background reference together with the bibliography it contains, which is complete up to 1965 and has been updated to 1974 by Wolf (1977).

It is well known that many different systems of many-valued logics have been presented and discussed. Many of the remarks in the paper will refer to a general many-valued system; very often, however, we shall focus our attention on the Łukasiewicz logics (often indicated by Ł). Presently, in fact, these are, among the existing many valued formal systems, the sounder and more sophisticated ones from a logical point of view.

In the following we shall briefly summarize and comment on some "classical" logical results of many valued logics (Section 1). Then, we shall briefly discuss the problem of vagueness in relation to many valued logics and the different attitudes one can have towards it (Section 2). Some views of ours on the problems presented by the analysis of complex systems and on the role played in it by the notions of vagueness and approximation will be briefly presented in Section 3. In the final Section some general epistemological remarks will be presented.

1. Logical Remarks.

We shall deal with two types of problems - "formal" ones (concerning the logical systems themselves) and "informal" ones (relating also to applications, e.g., to science) - and we want to stress immediately that these two groups of questions are strongly related to each other. Among the formal questions, we recall:

i) purely syntactical questions as, for instance, the existence of a Deduction Theorem;

ii) the search for an axiomatization complete with respect to a given semantics (and related questions).

The major syntactical properties of Łukasiewicz propositional systems - as, for instance, the Deduction Theorem - have been carefully investigated (see, e.g., Rosser and Turquette (1952), Rose and Rosser (1958), Guccione, Tortora and Vaccaro (1981), Guccione and Tortora (1982) and the bibliography by Porte (1982)).

Complete axiomatizations have been given for many of these systems: among the papers on this topic we recall Wajsberg (1935), Rosser and Turquette (1952), McNaughton (1951), Rose and Rosser (1958), Meredith (1958), Chang (1958), Rose (1953, 1968, 1978). (For a survey of results about other many valued systems, see, for instance, Rescher (1969), Ackermann (1971), Haack (1974)).

Let us observe that for propositional systems usually no serious semantical problems of *conceptual type* arise. The serious problems, when they arise, are technical ones. For propositional systems, in fact, it is enough to provide a quite elementary formal semantics: namely only a set of designated truth-values together with truth functions for the logical connectives. This is, of course, the natural extension of what is done in the Classical Propositional Calculus when the set {0,1} of truth values and the well known truth tables are provided. Łukasiewicz calculi play a central role, not only since the semantical definition of their primitive connectives is intuitively very natural and a huge number of positive results have been obtained for all the L propositional systems, but also because

47

interesting problems arise when studying their extensions to the first order predicate calculus. The main question is, of course, that of knowing whether there are sets of axioms (and of inference rules) such that a completeness theorem holds. The problem has been faced by many authors and the two limiting results have been obtained by Rutledge (1959) and Scarpellini (1963). The former has shown that Łukasiewicz's system restricted to monadic predicates is axiomatizable, whilst Scarpellini has shown that the full Ł is not axiomatizable in the specific sense that the set of the wffs of Ł that take always (in every model) the truth value 1 is not recursively enumerable. Intermediate results of *weak completeness* have been obtained by various authors. We remember here the papers by Belluce and Chang (1963), Hay (1963), Belluce (1964), Chang (1964). The most interesting result can be expressed (see, e.g., Belluce (1964)) as follows:

Let us call $r \in [0,1]$ a weak recursive real number (wrr) if and only if there exist two recursive functions ϕ and ψ such that $\psi(n) > 0$ and $\phi(n) < \psi(n)$ for all n, and $r = \inf_n (\phi(n) / \psi(n))$. Then, calling $D(r)$ the set of wffs of Ł whose truth values belong always to $[r,1]$, one has that $D(r)$ is or is not recursively enumerable respectively according to whether r is or is not a wrr.

Before moving to informal problems, we want to touch briefly on a question which in the literature of many valued logics is very often discussed but, unfortunately, often stated in a misleading way. The problem has to do with the validity of the principles of contradiction and excluded middle in many valued logics. Let us stress that the previous principles are purely *syntactical*. To state that in a formal system $\mathscr{L}$ the excluded middle holds is equivalent to say that $\vdash_{\mathscr{L}} \alpha \lor \sim\alpha$. To state that in a formal system $\mathscr{L}$ the contradiction principle holds is equivalent to say that $\vdash_{\mathscr{L}} \sim(\alpha \land \sim\alpha)$. The principle of excluded middle must not be confused with the so called *bivalence principle*, which on the contrary, is a *semantical* statement on the totality of the wffs of a system. The bivalence principle states that any wff is either true or false. Now it is clear that - by definition - the bivalence principle does not hold in any many valued logical system, but this does not imply that the principle of excluded middle cannot hold in any many valued logic. For instance in the system BL proposed by Sanford (1975) we have that even if *"propositions are admitted which have values between 0 and 1, no theorem of classical propositional logic fails"* (page 24). The real problem, then, is not to discuss, *in abstracto*, whether the principle of excluded middle (or of contradiction) fails in a certain many valued logic but whether it is possible to

provide, for this logic, a formal semantics corresponding to a useful informal interpretation (and then, ultimately, a domain of reality which is satisfactorily mapped by the given logical system). And this brings us to the second group of questions.

Let a many valued formal system be given. A critic of the many valued approach would certainly point out that the construction of such a system is nothing more than an initial move since, even if the system is "sound and sufficiently powerful" it is necessary to endow it with an "adequate" semantical apparatus. Suppose now that also this second step has been made, laying down a formal semantics (for instance, some algebraic semantics). This is not enough either: if the construction of a syntactic machinery is criticized for being not very significant, for the same fault one may criticize the couple < syntactic machinery, formal semantics > . The latter could be nothing more than a sophisticated game, relevant only within a restricted society of puzzle breakers interested in this particular game. It is necessary to provide an informal interpretation of the formal semantics in order to attach some *general* interest to the proposed logical system. The satisfaction of this requirement is just one of the major reasons of the usefulness and success of Tarski's and Kripke's semantics.

Well, it is just in relation to these questions that a weakness of many valued logics appears since the answers given to them are not considered satisfactory by many people. This is related both to the justification of the conceptual foundations of these logics and to the justification of their use in scientific theories. On these points see the challenging remarks made by D.Scott in the papers quoted in the bibliography. The title of the second of these papers, "Does Many-Valued Logic Have Any Use?", is truly the central question. We turn to it in the following section[2].

2. *Vagueness* and *fuzziness*.

The notion of *vagueness* will be looked at, here, as a semantical notion, following the point of view that has been expressed by Russell (1923), perhaps for the first time with clarity: words as "vague" *"have to do with the relation between a representation and that which it represents. Apart from representation ... there can be no such things as vagueness or precision; things are what they are"*. Roughly speaking, then, we can call vague all those predicates which have a field of application (better: an extension) intrinsecally doubtful. A vague statement, then, will be, for instance, one that contains occurrences of vague predicates. Let us immediately observe that what has been called "intrinsic doubtfulness" must not be confused either with the effective undecidability (in the

48

sense of the Theory of Recursive Functions) or with notions of probabilistic type or with questions of empirical (practical) undecidability.

Vagueness is in the crossroad of such notions as imprecision of a measure, imprecision of a model, verisimilitude, partial truth, imprecision in the description of complex systems, fuzziness and so on. This is not the place to try to draw a map of the correlations among these various concepts and notions. However, we want to stress that the problems that have been previously mentioned are part of the epistemological debate on the role that precision and imprecision (or vagueness) can play (or, better, do play) in science, particularly in relation to the notions of explication and complexity of a model. All these problems have, perhaps, their *focus* in the so called *Carnap-Popper controversy* in which Carnap's position is characterized by a total rejection of vague expressions in scientific language (see, e.g., Carnap (1950), Ch.1) whereas Popper is, at least, more tolerant (see, e.g., Popper (1976)).

One could then ask whether it is better to work on a program of systematic elimination (Eliminating Program) of every vague expression from the scientific discourse or to outline a research program able to cope with the presence of vague expressions by controlling the presence and the use of vagueness (Controlling Program). A very interesting approach following the path of the Eliminating Program is, for instance, the one proposed by Fine (1975). In the following, however, we shall concentrate on works that follow the path of the Controlling Program.

We first note that if one does not follow the point of view of the Eliminating Program, a preliminary problem is the one of finding an adequate logical way of treating vagueness. The presence of vague predicates, in fact, is not always compatible with a straightforward use of the connectives of classical logic (see, e.g., Black (1937)). Many authors agree, as we do, that many valued logic seems an adequate tool - in particular a logic taking the interval [0,1] of the real line as the set of truth values[3]. In particular -notwithstanding the limitative theorems mentioned in the previous section - Łukasiewicz predicate calculus is widely considered as the most adequate candidate to the role of *Calculus of Vague Predicates*.

Among those who have criticized the use of L are Morgan and Pellettier (1977), whereas Machina (1976) has argued in its favour. We think that Machina's paper is very interesting in the direction previously suggested: among other reasons, since he proposes a set-theoretic semantics for *Ł*. This semantics is developed in the setting of fuzzy set theory (Zadeh, 1965). Another interesting paper which relates Łukasiewicz's logic to fuzzy sets is

due to R. Giles (1975). The line developed by Giles is, however, different. In order to develop a language suitable to the formalization of physical theories, he proposes a non classical logic (defined by means of a dialogue interpretation) which reduces - under some further reasonable assumptions - to the infinite valued *Ł* (Giles, 1974). Giles then in his (1975) constructs a theory of fuzzy sets starting from and in terms of this new logic. The relationship between them is well expressed by his own words: "*indeed it is not too much to claim that Ł is related to fuzzy set theory exactly as classical logic is related to ordinary set theory*". Giles, finally, in this same paper quotes a neglected series of papers (in German) by Klaua "*who develops a many valued set theory based on Łukasiewicz logic in a manner similar to (but much more sophisticated than) that adopted*" in his own paper.

These last remarks have brought us to two knots of tangled and difficult problems. The first is the sublabyrinth of "set theories" adequate to corresponding many valued logics. The second is the sublabyrinth of the so called *fuzzy logics*[4]. For what regards the former it is outside the aim of the present paper and we refer the reader to the references already quoted and also to Klaua (1966b) and to Gottwald (1976). The second sublabyrinth would also deserve a paper in its own. We shall limit ourselves here to the following brief remarks. The original motivation behind Łukasiewicz many valued logics was of a general philosophical type and only later (see,e.g., the papers by Scott and Katz quoted in the bibliography) the idea arose of using them as theoretical tools for dealing with imprecision, error and approximation. Fuzzy logics, instead, from the start has been characterized as the best candidates for the modelling of inexact, approximate reasoning. Their *rationale*, in fact, as well as the one of fuzzy sets was just to provide tools - set theoretical in the latter case, logical in the former - for dealing immediately with situations whose description was strongly characterized by fuzziness and approximation. A forerunner of the idea of using notions and tools from fuzzy set thery for logically dealing with inexact concepts is J.A. Goguen (1969). A very interesting review is due to B.R. Gaines (1976).

The name Fuzzy Logics, however, has to be specifically associated with the proposal by L.A. Zadeh (1975a, 1975b) and R. Bellmann and L.A.Zadeh (1977). There are many innovative points in this program which is really very challenging and new. For instance, the truth values are fuzzy sets representing notions like *true, very true, more or less true*, etc. The truth values then are not numerical but linguistic; moreover they may be generated by a grammar and interpreted by a semantical rule. The connectives may have a variable meaning

49

and the inference rules are approximate. However if the program is taken in its full generality it is very difficult both to evaluate by means of classical standards the results obtained and to understand clearly which generalizations are really achieved with respect to classical theories. For some criticism of fuzzy reasoning see the paper by Schefe (1981) and for constructive suggestions for further developments see Skala (1982). Other papers presented also under the heading of "fuzzy logics" can be seen, more specifically, as extensions of Łukasiewicz logics. Among them we can put Pavelka (1979), Albert (1977) and Gottwald (1980).

We have finally to mention two wide groups of people which could be grouped under the names of "British School" and "Barcelona School". For a sample of the work done by the first group see, for instance, Baldwin (1979), Baldwin and Pilsworth (1978), Bandler and Kohout (1980), Kohout and Bandler (1982), Willmott (1981). The main aim of these works has been a careful study of many different logical connectives bearing in mind not only their theoretical interest but also possible applications (which go from the design of industrial plants to medical diagnosis). The Barcelona School centered around Enric Trillas has undertaken a complementary work, namely the characterization of classes of connectives satisfying certain requirements and which are functionally expressable[5]. See Trillas (1982), Trillas, Alsina and Valverde (1982), Trillas, Domingo and Valverde (1981), Trillas and Valverde (1981) and Esteva (1981).

We want finally to mention that extensive work has been done in applying many of these new ideas born out in many valued and fuzzy logics to switching theory. The Proceedings of the previous ISMVL's are the best general reference. Let us specifically mention Mukaidono's study and characterization of "canonical forms" of fuzzy switching functions (see, e.g., his 1980 and the references therein).

3. *Complex Systems and Approximation.*

The above relatively long excursus of the relationship between many valued logics and vagueness and/or fuzziness had not only logical and epistemological aims. The main aim was to wind off our Ariadne's thread through the labyrinth of many valued logics in order to reach the *locus* of Complex Systems. For simplicity we shall refer here to *artificial* complex systems. We then assume (leaving out sophisticated epistemological investigations) that the distinction natural/artificial makes sense and interpret it in a naive and intuitive manner.

In our opinion, the admission of fuzziness and vagueness in the modelling of a system, *at a formal level*, has to be justified only on the basis of the possibility of obtaining, in this way, models

of the given system which are better and better *from the point of view of handling and explicative power*. We do not share, then, the position of those supporters of fuzzy models and fuzzy logics who state that the usefulness of these logics and models springs out from the fact that *the real world is pervaded with approximation and imprecision*.

Vagueness and fuzziness are semantical notions that come out from a discrepancy between "a representation and what it represents", to quote again Russell. Fuzziness and vagueness are then measures of how much a model maps, or does not map, a certain real system or a certain piece of reality (better, the *information* we have on that system or on (that part of) reality). In principle, the best model remains a *crisp, non fuzzy* one, but the crispness has to be the final goal, to be achieved as a result of a real and complete correspondence between the model and the system and not by ad hoc and possibly arbitrary oversimplification of the system. Two main cases that we want to provide as examples of our thesis are the following:
A) Let us suppose that in studying a complex system we are mainly interested in its sophisticated behaviour. Our aim is then to obtain a model of the system as manageable as possible but still able to mirror this behaviour. Models mapping with extreme precision all the real system could be - at the present level of knowledge - too difficult to handle from a mathematical point of view. But even if this would be possible, the price to be paid (measured by some parameter) would be too high relative to the specific purposes.
B) Let us now consider the case in which the real system under study is highly complex so that it is not possible - at the present level of knowledge - to provide a crisp and complete model on the basis of the information actually available. In some cases it could even be dangerous from the point of view of the *explication* to provide a supersimplified model of the system (even if crisp). Such a model, in fact, could fail to represent the real system even under the subsequent addition of further hypotheses (think, for instance, of systems in which non linearity plays an essential role *versus* linear models - however sophisticated - of the same system).

In both cases A and B, vague or fuzzy models could represent, then, *descriptive modalities* of the considered systems more adequate to our requirements not only at a *practical* level but also at the level of *explication*. Let us stress, however, that this is valid only under the additional condition that the inexactness (vagueness, fuzziness) introduced in the model is *controllable with precision at a meta-level*. This means that we should have precise formal explicata of vagueness and fuzziness which one can manage according to the usual and

standard procedures considered scientifically acceptable. For what regards our topic, this means to have many valued logical structures adequate for this purpose. We have already mentioned that the weak completeness results of first order L could be usefully interpreted in this direction. Other relevant and interesting recent results are the ones obtained by Pavelka (1979), which also reinforce the central role played by Łukasiewicz systems.

The leading idea at the root of the present paper is, then, that a *controlled quantity of inexactness* can be useful and suitable not only for a technical study and analysis of some classes of systems but also for their *description/explication* (and so *understanding*) at various levels of complexity. Fundamentally, this is nothing more than a different way of approaching the *notion of approximation* which plays a central role in scientific methodology. Even if this path is different from the classical one, it is not meant to be an alternative but to complement and to play its role in those cases - like the ones outlined - in which it seems more suitable[6].

The usefulness of facing directly the notion of approximation has been acknowledged in pure mathematics. M. Rabin (1976), e.g., has proposed an interesting notion of "proof" with a certain margin of error, to be used in situations in which an exact proof would be unattainable.

Let us finally mention the challenging topic of considering many valued logic straightforwardly as logic of approximation. We limit ourselves to stressing the interesting work that M. Katz is doing in a series of papers (see, e.g., his 1981 a and b, 1982). In his 1981 b the author develops a semantical analysis of two first order real valued logics. The first (the logic of inexactness) is mainly Scott's version of Łukasiewicz logic, the second (the logic of approximation) is just an attempt to represent mathematically the notion of approaching the truth as near as one wishes. From a mathematical point of view the main results of this logic can be found in his (1982). Finally, in the paper by Guccione and Tortora (1982) the idea of interpreting many valued logic as logic of approximation is related to the new notion of *levels of provability*.

Concluding Remarks.

As we stated in advance, this note aims at no completeness, either about the logical and philosophical problems of many valued logics or about the wellfoundedness of their several applications.

Let us then briefly conclude stressing a central problem of logical nature: the possibility of translating a logical system into another one. Unfortunately, there is little work in this field:

for what regards many valued logics we want to mention - among the few papers we know - Woodruff (1974) and Duff (1979).

We think that further analyses of these problems would be very useful both for the understanding of non classical logical structures and for having a good guide to their applications.

FOOTNOTES

1. The title of the present paper - as well as of the analogous review of Dialectical Logics (Guccione and Tamburrini, 1982) - is inspired by the well known essay "The Labyrinth of Quantum Logics" by Bas van Fraassen (1974). Let us incidentally note a main difference that immediately comes out between Quantum Logics on one side and Many Valued Logics (and Dialectical Logics) on the other side. Quantum Logics - notwithstanding their possible drawbacks - have their referent and are rooted in Quantum Theory, which in its turn - notwithstanding its (minor) drawbacks - is rooted in the large class of empirical facts of the microscopical world it helps to understand. The situation is completely different for many valued logics (and worse for dialectical logics). We are deeply convinced that further developments and applications will show their central importance in all those problems strongly characterized by approximation and incomplete description. However at present it is very difficult to pin out theories (and related domains of empirical phenomena) which show a relationship *with many valued logics comparable to the one existing between Quantum Theory and Quantum Logics.* (For a counter example, however, see the papers by Skala (1978), Ovchinnikov (1982) and Nurmi (1982).

2. A relevant point not mentioned in the present survey is the problem of identity in many valued logics. See, for instance, Thiele (1958), Gottwald (1983) and - in the context of fuzzy sets - Pultr (1982).

3. Let us recall, for instance, a modal (propositional) system with infinitely many truth values, by Sanford (1975). See also Guccione and Termini (1979) for brief comments on this system and the outline of an alternative proposal.

4. For an interesting analysis of the various ways of using the term "fuzzy logic" see the comprehensive survey by Gottwald (1981).

5. In the setting of the work done in Barcelona it has to be mentioned also the recent doctoral dissertation by Ton Sales (1982) which looks at many valued logics (in their most general sense) as a tool for the logical analysis of imprecision.

6. Among the possible fields of application let us mention Social Sciences. Without entering problems

of evaluation of the results obtained in this area by means of many valued logical structures or of fuzzy models let us refer to the papers by Skala, Ovchinnikov and Nurmi quoted in footnote 1.

Bibliography

Ackermann, R. (1971) - An Introduction to Many-valued Logics. Routledge and Kegan Paul.

Albert, P. (1977) - The Algebra of Fuzzy Logic. Fuzzy Sets and Systems, 1, 203-230.

Baldwin, J.F. (1979) - A new Approach to Approximate Reasoning using a Fuzzy Logic. Fuzzy Sets and Systems, 2, 309-326.

Baldwin, J.F. and B.W. Pilsworth (1978) - Axiomatic Approach to Implication for Approximate Reasoning with Fuzzy Logics. Fuzzy Sets and Systems 3, 193-220.

Bandler, W.C. and L.J. Kohout (1980) - Semantics of Implication Operators and Fuzzy Relational Products. Int. J. Man-Machine Studies 12, 89-116.

Belluce, L.P. (1964) - Further Results on Infinite-Valued Predicate Logic. J.S.L. 29, 69-78.

Belluce, L.P. and C.C. Chang (1963) - A Weak Completeness Theorem for Infinite Valued First Order Logic. J.S.L. 28, 43-50.

Bellmann, R.E. and L.A. Zadeh (1977) - Local and Fuzzy Logics. In J.M. Dunn and G. Epstein (eds) Modern Uses of Multiple-Valued Logics, 103-165. D. Reidel.

Black, M. (1937) - Vagueness: An Exercise in Logical Analysis. Phil. of Science 4, 427-455.

Carnap, R. (1950) - Logical Foundations of Probability. University of Chicago Press.

Chang, C.C. (1958) - Proof of an Axiom of Łukasiewicz. Trans.Am.Math.Soc. 87, 55-56.

Chang, C.C. (1964) - Infinite Valued Logic as a Basis for Set Theory. In Y. Bar-Hillel (ed.), Proc.of 1964 Int.Congr. for Logic, Methodology and Phil. of Science, Amsterdam, 93-100.

Duff, M.J. (1979) - Modal Interpretation of Three-Valued Logics (I and II). Notre Dame J.of Formal Logic, 20.

Esteva, F. (1981) - On the Form of Negations in Posets. Proc. of the 11th ISMVL, Oklahoma City, 228-231.

Fine, K. (1975) - Vagueness, Truth and Logic. Synthese, 30, 265-300.

Gaines, B.R. (1976) - Foundations of Fuzzy Reasoning. Int. J. of Man-Machine Studies 8, 623-668.

Giles, R. (1974) - A Nonclassical Logic for Physics. Studia Logica 33, 397-415.

Giles, R. (1975) - Łukasiewicz Logic and Fuzzy-set Theory. Proc. of the 5th ISMVL, 197-211. Also in Int. J. of Man-Machine Studies 8 (1976), 313-327.

Goguen, J.A. (1969) - The Logic of Inexact Concepts. Synthese 19, 325-373.

Gottwald, S. (1976) - Untersuchungen zur Mehrwertigen Mengenlehre. Math.Nachr. I: 72, 297-303; II: 74, 329-336; III: 79, 207-217.

Gottwald, S. (1980) - Fuzzy Propositional Logics. Fuzzy Sets and Systems 3, 181-192.

Gottwald, S. (1981) - Fuzzy-Mengen und ihre Anwendungen. Ein Überblick. E.I.K. 17, 207-235.

Gottwald, S. (1983) - Fuzzy Set Theory, Some Aspects of the Early Development. In Skala et al.(eds.), Aspects of Vagueness, D.Reidel (to appear).

Guccione, S. and G. Tamburrini (1982) - The Labyrinth of Dialectical Logics. Proc. of the 2nd World Conf. on Math. at the Service of Man, 318-321.

Guccione, S. and S. Termini (1979) - The Modal "Vaguely". Proc. of the 6th Int.Congr.of Logic, Methodology and Phil.of Science, 5, 76-80.

Guccione, S. and R. Tortora (1982) - Deducibility in Many-Valued Logics. Proc. of the 12th ISMVL, Paris, 117-121.

Guccione, S., R. Tortora and V. Vaccaro (1981) - Deduction Theorems in Łukasiewicz Propositional Calculi. Rendiconti Sem. Mat. Univ. e Politecn. di Torino 39, 53-65.

Haack, S. (1974) - Deviant Logic. Cambridge Un.Pre.

Hay, L.S. (1963) - Axiomatization of the Infinite-Valued Predicate Calculus. J.S.L. 28, 77-86.

Katz, M. (1981) - Łukasiewicz Logic and the Foundations of Measurement. Studia Logica 40, 209-225.

Katz, M. (1981) - Two Systems of Multi-Valued Logic for Science. Proc. of the 11th ISMVL, 175-182.

Katz, M. (1982) - Real-Valued Models with Metric Equality and Uniformly Continuous Predicates. J.S.L. (to appear).

Klaua, D. (1966) - Über einen zweiten Ansatz zur mehrwertigen Mengenlehre. Monatsb.Deutsch Akad.Wiss.

Berlin, 8, 161-177.

Klaua, D. (1966 b) - Grundbegriffe einer Mehrwertig-
en Mengenlehre. Monatsb. Deutsch Akad. Wiss. 8, 782-
802.

Kohout, L.J. and W. Bandler (1982) - Axioms for Con-
ditional Inference: Probabilistic and Possibilistic.
Proc. 2nd World Conf. on Math. at the Service of
Man. Las Palmas, 413-414.

Machina, K.F. (1976) - Truth, Belief and Vagueness.
Journal of Phil. Logic 5, 47-78.

McNaughton, R. (1951) - A theorem About Infinite
Valued Sentential Logic. J.S.L. 16, 1-13.

Meredith, C.A. (1958) - The Dependence of an Axiom
of Łukasiewicz. Trans. A.M.S. 87, 54.

Morgan, C.G. and F.J. Pellettier (1977) - Some Notes
Concerning Fuzzy Logics. Linguistic and Philosophy
1, 79-97.

Mukaidono, M. (1982) - New Canonical Forms of Fuzzy
Switching Functions. Proc. 2nd World Conf. on Math.
at the Service of Man. Las Palmas, 520-523.

Nurmi, H. (1982) - The resolution of Allais Paradox.
Stochastica (to appear).

Ovchinnikov, S. (1982) - Social Choice and Łukasie-
wicz Logic. Proc. 12th ISMVL (Paris), 163-166.

Pavelka, J. (1979) - On Fuzzy Logic. Zeitsch. math.
Logik und Grundlagen Math. 25, 45-52,119-134,447-464.

Popper, K. - Unended Quest. London, 447-464.

Porte, J. (1982) - Fifty years of Deduction Theorem.
Proc. Herbrand Symposium, Logic Colloquim '81,
J. Stern (ed.), North Holland.

Pultr, A. (1982) - Fuzziness and Fuzzy Equality.
Comment. Math. Univ. Carolinae 23, 249-267.

Rabin, M.O. (1976) - Probabilistic Algorithms. In
J.F. Traub (ed), Algorithms and Complexity, 21-40.

Rescher, N. (1969) - Many-Valued Logic, New York.

Rose, A. (1953) - A Formalization of an $\aleph_o$- valued
propositional calculus. Proc. Cambridge Phil. Soc.

Rose, A. (1968) - Formalization of some $\aleph_o$ - valued
Łukasiewicz propositional calculi. Lecture Notes on
Math., Springer-Verlag, vol. 70, 269-271.

Rose, A. (1978) - Formalization of further $\aleph_o$-valued
Łukasiewicz propositional calculi. J.S.L.43,207-210.

Rose, A. and Rosser, J.B. (1958) - Fragments of many
valued statement calculi. Trans. A.M.S., 87, 1-53.

Rosser, J.B. and Turquette, A.R. (1952) - Many-valued
Logics, North Holland.

Russell, B. (1923) - Vagueness. Australian Jour. of
Philosophy. 1, 84-92.

Rutledge, J.D. (1959) - A preliminary Inv. of the In-
finite Valued Pred. Logic.Ph.D. Th. Cornell Univers.

Sales, T. (1982) - Contr.to the logical anal. Impre-
cision (catalan), Ph.D. Th. Polyt. Univ. Barcelona.

Sanford, D.H. (1975) - Borderline Logic. Am. Phil.
Quarterly. 12, 29-39.

Scarpellini, B. (1963) - Die Nichtaxiom. der unendlich.
Logik von Łukasiewicz-Tarski. J.S.L. 27, 19-22.

Schefe, P. (1981) - On found. of reasoning with uncer-
tain facts and vague concepts. In Mamdani, Gaines (eds)
Fuzzy Reasoning and its Appl., London, 189-216.

Scott, D.S. (1974) - Completeness and Axiomatizabil.
in Many Valued Logic. Proc. Symp. Pure Math.25,188-197.

Scott, D.S. (1976) - Does Many Valued Logic have Any
Use? In Körner (ed),Philosophy of Logic, 64-74.

Skala, H.J. (1978) - Arrow's Impossibility Theorem.In
Gottinger and Leinfellner (eds),Decision Theory and
Social Ethics. D. Reidel, 215-226.

Skala, H.J. (1982)-Modelling Vagueness,in Gupta and
Sanchez (eds) Fuzzy Inf.and Decision Proc. 101-109.

Thiele, H. (1958)-Theorie endlich. Łukasiewiczschen
Prädik. Ersten St. Zeitsch.math.Logik 4, 108-142.

Trillas, E.(1982)-Indistinguishability relations (in
catalan) Proc. First Catalan Congr. Mathem. Logic.

Trillas, E., C. Alsina and Ll. Valverde(1982)-Do we
need max, min and 1-j in Fuzzy set Theory? in Yager
(ed) Recent Develop.of Fuzzy Set and Possibility Th.

Trillas, E., X.Domingo and Ll.Valverde (1981)-Pushing
Łukasiewicz implic. little further. 11 ISMVL, 232-234.

Trillas,E. and Ll. Valverde (1981)-Functional Implic.
in FST. in Klement (ed) 3rd Sem. Fuzzy sets, 173-190.

van Fraassen, B.C. (1974)-The Labyrinth of Quantum
Logics. B.S.P.S. 13, 224-254.

Wajsberg, M. (1935)-Beitrage zum metaussagenkalkul I.
Monatshefte für Math. und Physik. 42, 221-242.

Willmott, R. (1981)- Mean measures containment and
equality betw. fuzzy sets. Proc. 11th ISMVL, 183-190.

Wolf, R.G. (1975)-Crit. Survey Many-val.Log.1966-74.
In Dunn, Epstein (eds),Modern uses M.V.Log. 167-323.

Woodruff, P.W. (1974)-A Modal interpret. of 3-valued
Logic. J. Phil. Logic 3, 433-439.

Zadeh, L.A.(1965)-Fuzzy sets. Inform. Contr. 8,338-353.

Zadeh, L.A.(1975a)- Fuzzy Logic and Approximate
Reasoning. Synthese 30, 407-428.

Zadeh, L.A.(1975b)- The Concept of Linguistic varia-
ble and its Application to Approximate Reasoning.
Inf. Sc. 8, 199-249, 301-357; 9, 43-80.

Sesion 2A
Circuit and Technology I

AD P002332

THE NEW METHOD OF IMPLEMENTATION FOR TERNARY LOGIC SYSTEM

LI Meng   GU Wei-Nan

Department of Telecommunication,
Shanghai Institute of Railway Technology, Shanghai, China.

Abstract

In various ternary logic systems, symmetric ternary logic system which satisfies lattice operation is most appropriate. A new method of implementation for ternary logic using COS/MOS circuits is put forward, only one circuit simple structure can constructs a ternary logic complete system.

The paper also discusses the possibility of applying this circuit for the implementation of J-OR and J-AND operation. Thus, the implementation for ternary logic system can be greatly simplified.

Introduction

A conventional ternary logic system which satisfies lattice operation constructs a complete system with at least general J-gates and NAND gates. Any ternary function may be expressed as follows in a lattice operation form:

$$F = \sum_{f(a_1,a_2,\ldots,a_n)=1}(\prod J_m(x_i)) + 0 \cdot \sum_{f(a_1,a_2,\ldots,a_n)\in\{0,1\}}(\prod J_m(x_i))$$

$$m\in\{\bar{1},0,1\} \qquad i\in\{1,2,\ldots,n\}$$

It is obvious that lattice operation needs a large number of J-gates and NAND gates, thus making logic design circuits very complex and expensive. In addition to the complexity of the gates structure and the low noise allowance, it is difficult for the ternary logic system to compete with binary logic system.

A new circuit is put forward in the paper, This circuit of two outputs will construct a complete system its own. Many new simplifying methods developed in the paper may be used, so that the lattice operation system will be greatly simplified.

II. Definition of symmetric ternary logic system satisfying lattice operation

Symmetric ternary logic system which satisfies lattice operation possesses more advantages than other ternary logic system. Hence, this system is used in the paper.

Logic value are $\bar{1},0,1$. Logic value satisfies the relation: $\bar{1} < 0 < 1$.

The definitions of binary operations are:

sum operation $x + y = \max(x,y);$
$$x,y\in\{\bar{1},0,1\}$$
product operation $x \cdot y = \min(x,y);$
$$x,y\in\{\bar{1},0,1\}$$

There are more one variable operation, these operation possessed their own circuits. But only some operations must be considered, the others could not be considered in analysis.

J- operation will be replaced by J-AND operation. J-gate can be replaced by J-AND gate. All these definitions are described as follow:

Complement operation:

$$\bar{x} = \begin{cases} \bar{1} \;; & \text{if} \quad x = 1 \\ 0 \;; & \text{if} \quad x = 0 \\ 1 \;; & \text{if} \quad x = \bar{1} \end{cases}$$

Right circular operation:

$$\overrightarrow{x} = \begin{cases} 1 \;; & \text{if} \quad x = 0 \\ 0 \;; & \text{if} \quad x = \bar{1} \\ \bar{1} \;; & \text{if} \quad x = 1 \end{cases}$$

Left circular operation:

$$\overleftarrow{x} = \begin{cases} \bar{1} \;; & \text{if} \quad x = 0 \\ 0 \;; & \text{if} \quad x = 1 \\ 1 \;; & \text{if} \quad x = \bar{1} \end{cases}$$

J-operation may be discribed as a truth table below:

$J(x)$ / x	$J_{\bar{1}}(x)$	$J_0(x)$	$J_1(x)$	$J_{\bar{1}0}(x)$	$J_{01}(x)$	$J_{\bar{1}1}(x)$
$\bar{1}$	1	$\bar{1}$	$\bar{1}$	1	$\bar{1}$	1
0	$\bar{1}$	1	$\bar{1}$	1	1	$\bar{1}$
1	$\bar{1}$	$\bar{1}$	1	$\bar{1}$	1	1

J-AND operation or J-OR operation corresponds to a ternary logic miniterm. First, make product operation or sum operation; then do J- operation.

Definitions of J-AND operation and J-OR are as follow:

$$J_m\left(\sum x_i\right) = \begin{cases} 1; & \text{if } \prod x_i \in m \\ \bar{1}; & \text{if } \prod x_i \notin m \end{cases}$$

$$J_m\left(\sum x_i\right) = \begin{cases} 1; & \text{if } \prod x_i \in m \\ \bar{1}; & \text{if } \prod x_i \notin m \end{cases}$$

$\sum$ is continual sum symbol. $\prod$ is continual product symbol.

$$m \in \{\bar{1}, 0, 1, 01, \bar{1}1, \bar{1}0\}$$
$$x_i \in \{x_1, x_2, \ldots, x_n\}$$

The general expression of symmetric ternary logic function satisfying lattice operation is

$$F = f(a_1, a_2, \ldots, a_n) \sum \left(\prod J_m(a_i)\right) + 0 \cdot f(a_1, a_2, \ldots, a_n) \sum \left(\prod J_m(a_i)\right)$$

where $f(a_1, a_2, \ldots, a_n) = 1$ are called 1-implicant. where $f(a_1, a_2, \ldots, a_n) = 0$ are called 0-implicant. 1-implicant is called don't care term of 0-implicant.

Any ternary logic function will be expressed and its completiness has been proven by many authors.

According to this expression, any ternary function can be implemented by using J-AND gate and NAND gate. Therefore, J-AND gate and NAND are complete.

III. Simplifying Rules

J-operation properties are as follow:

1. J-operation with same variables may be merged.

(1)　$J_{\bar{1}}(x) + J_1(x) = J_{\bar{1}1}(x) = \overline{J_0(x)}$

(2)　$J_0(x) + J_1(x) = J_{01}(x) = \overline{J_{\bar{1}}(x)}$

(3)　$J_{\bar{1}}(x) + J_0(x) = J_{\bar{1}0}(x) = \overline{J_1(x)}$

2. Each J-operation can be transfered with other:

(1)　$J_1(x) = J_{\bar{1}}(\bar{x})$

(2)　$J_0(x) = J_{01}(x \cdot \bar{x}) = J_{\bar{1}0}(x + \bar{x})$
$$= \overline{J_{\bar{1}}(x \cdot \bar{x})}$$

where $x \in \{x_1, x_2, \ldots, x_n; \cdot, +\}$.

These simplifying methods are very easy to proven. We can prove it by iterative method and by truth table. The generalised rules of n-variable can also be proven by inductive method.

According to J-operation's mergerable property, ternary logic function expression can be simplified easily. And according to J-operation transferable property, J-gate and NAND-gate will construct complete system. The structure of J-gate and NAND-gate is very simple, but the structure of J-gate will become complex.

J-AND operation's simplifying rules are described as below:

1. $J_1(x) \cdot J_1(y) = J_1(x \cdot y)$
1'. $J_1(x_1) \cdot J_1(x_2) \cdot \cdots \cdot J_1(x_n) = J_1\left(\prod_{i=1}^{n} x_i\right)$
2. $J_1(x) \cdot J_{\bar{1}}(y) = J_{\bar{1}}(x+y)$
2'. $J_{\bar{1}}(x_1) \cdot J_{\bar{1}}(x_2) \cdot \cdots \cdot J_{\bar{1}}(x_n) = J_{\bar{1}}\left(\sum_{i=1}^{n} x_i\right)$
3. $J_0(x) \cdot J_0(y) = J_{01}(x \cdot \bar{x} \cdot y \cdot \bar{y})$
3'. $J_0(x_1) \cdot J_0(x_2) \cdot \cdots \cdot J_0(x_n) = J_{01}\left(\prod_{i=1}^{n} x_i \bar{x}_i\right)$
4. $J_{01}(x) \cdot J_{01}(y) = J_{01}(x \cdot y)$
4'. $J_{01}(x_1) \cdot J_{01}(x_2) \cdot \cdots \cdot J_{01}(x_n) = J_{01}\left(\prod_{i=1}^{n} x_i\right)$
5. $J_{\bar{1}0}(x) \cdot J_{\bar{1}0}(y) = J_{\bar{1}0}(x+y)$
5'. $J_{\bar{1}0}(x_1) \cdot J_{\bar{1}0}(x_2) \cdot \cdots \cdot J_{\bar{1}0}(x_n) = J_{\bar{1}0}\left(\sum_{i=1}^{n} x_i\right)$
6. $J_{\bar{1}1}(x) \cdot J_{\bar{1}1}(y) = J_{\bar{1}1}(x \cdot \bar{x} + y \cdot \bar{y})$
$$= J_{\bar{1}}((x+\bar{x}) \cdot (y+\bar{y}))$$
6'. $J_{\bar{1}1}(x_1) \cdot \cdots \cdot J_{\bar{1}1}(x_n) = J_{\bar{1}}\left(\sum_{i=1}^{n} x_i \bar{x}_i\right)$
$$= J_1\left(\prod_{i=1}^{n} (x_i + \bar{x}_i)\right)$$
7. $J_1(x) \cdot J_{\bar{1}}(y) = J_1(x \cdot \bar{y})$
8. $J_{01}(x) \cdot J_{\bar{1}0}(y) = J_{01}(x \cdot \bar{y})$
9. $J_0(x) \cdot J_{01}(y) = J_{01}(x \cdot y \cdot \bar{x})$
10. $J_0(x) \cdot J_{\bar{1}0}(y) = J_{01}(x \cdot \bar{x} \cdot \bar{y})$

J-OR operation's simplifying rules are as following:

1. $J_1(x) + J_1(y) = J_1(x+y)$
1'. $J_1(x_1) + J_1(x_2) + \cdots + J_1(x_n) = J_1\left(\sum_{i=1}^{n} x_i\right)$
2. $J_{\bar{1}}(x) + J_{\bar{1}}(y) = J_{\bar{1}}(x \cdot y)$

$2'.\quad J_{\overline{1}}(x_1)+J_{\overline{1}}(x_2)+\cdots+J_{\overline{1}}(x_n)=J_{\overline{1}}\left(\prod_{i=1}^{n}x_i\right)$

$3.\quad J_0(x)+J_0(y)=J_{01}(x\cdot\overline{x}+y\cdot\overline{y})$

$3'.\quad J_0(x_1)+J_0(x_2)+\cdots+J_0(x_n)=J_{01}\left(\sum_{i=1}^{n}x_i\overline{x_i}\right)$

$4.\quad J_{01}(x)+J_{01}(y)=J_{01}(x+y)$

$4'.\quad J_{01}(x_1)+J_{01}(x_2)+\cdots+J_{01}(x_n)=$
$\qquad\qquad =J_{01}\left(\sum_{i=1}^{n}x_i\right)$

$5.\quad J_{\overline{1}0}(x)+J_{\overline{1}0}(y)=J_{\overline{1}0}(x\cdot y)$

$5'.\quad J_{\overline{1}0}(x_1)+J_{\overline{1}0}(x_2)+\cdots+J_{\overline{1}0}(x_n)=$
$\qquad\qquad =J_{\overline{1}0}\left(\prod_{i=1}^{n}x_i\right)$

$6.\quad J_{\overline{1}1}(x)+J_{\overline{1}1}(y)=J_1(x+\overline{x}+y+\overline{y})$

$6'.\quad J_{\overline{1}1}(x_1)+J_{\overline{1}1}(x_2)+\cdots+J_{\overline{1}1}(x_n)=$
$\qquad\qquad =J_1\left(\sum_{i=1}^{n}(x_i+\overline{x_i})\right)$

$7.\quad J_1(x)+J_{\overline{1}}(y)=J_1(x+\overline{y})$

$8.\quad J_{01}(x)+J_{\overline{1}0}(y)=J_{01}(x+\overline{y})$

$9.\quad J_0(x)+J_{01}(y)=J_{\overline{1}0}(x\cdot\overline{x}\cdot\overline{y})$

$10.\quad J_0(y)+J_{\overline{1}0}(x)=J_{\overline{1}0}(x\cdot y\cdot\overline{y})$

These simplifying rules are very easy to be proven. We can prove it by iterative method and truth table.

The general simplifying form is discussed below. Every ternary logic function is always expressed as sum of 1-implicant and 0-implicant, and any miniterm among those is product form of J-operation.

$$f = \prod_{i=1}^{n}J_m(x_i); \quad i\in\{1,2,\ldots,n\}$$
$$m\in\{1,0,\overline{1},\overline{1}0,01,\overline{1}1\}.$$

If i is not appeared in the above expression, then $J_{\overline{1}01}(x_i)=1$ could be omitted.

According to J-operation's properties and simplifying rules, we may obtain a general expression as below:

$$f = J_1\Big(J_1\big(\prod_{J_1(x_p)=1}x_p\cdot\prod_{J_{\overline{1}}(x_q)=1}\overline{x_q}\big)\cdot$$
$$\cdot J_{01}\big(\prod_{J_0(x_r)=1}x_r\cdot\overline{x_r}\big)\Big)$$

$$p,q,r\in\{1,2,\ldots,n\}$$

From this expression, we could see: a minimum term corresponds to a multi-output J-AND-OR gate, and it is not a NAND gate operated by J-operation. In following circuit discussion, one may see that this complicated expression corresponds to a cheaper and simpler circuit. Because the characteristics of J-AND gate is better than NAND gate, we could replace NAND gate with J-AND gate and J-OR gate as much as possible. Thus, we could use least numbers of NAND gate to implement any ternary logic function and the number of NAND input is least.

III. Examples

We shall make use of examples to demonstrate the merits by using former method, and try to analyse these.

Example 1. Implement decorder circuits

Let x,y,z are three Flip-Flop's states:

The implementation for decoder circuit is given as following:

x	y	z	output
$\overline{1}$	$\overline{1}$	$\overline{1}$	$J_1(\overline{x}\cdot\overline{y}\cdot\overline{z})$
$\overline{1}$	$\overline{1}$	0	$J_1(\overline{x}\cdot\overline{y}\cdot J_{01}(z\cdot\overline{z}))$
$\overline{1}$	$\overline{1}$	1	$J_1(\overline{x}\cdot\overline{y}\cdot z)$
$\overline{1}$	0	$\overline{1}$	$J_1(\overline{x}\cdot\overline{z}\cdot J_{01}(y\cdot\overline{y}))$
$\overline{1}$	0	0	$J_1(\overline{x}\cdot J_{01}(y\cdot\overline{y})\cdot J_{01}(z\cdot\overline{z}))$
$\overline{1}$	0	1	$J_1(\overline{x}\cdot z\cdot J_{01}(y\cdot\overline{y}))$
$\overline{1}$	1	$\overline{1}$	$J_1(\overline{x}\cdot y\cdot\overline{z})$
$\overline{1}$	1	0	$J_1(\overline{x}\cdot y\cdot J_{01}(z\cdot\overline{z}))$
$\overline{1}$	1	1	$J_1(\overline{x}\cdot y\cdot z)$
0	$\overline{1}$	$\overline{1}$	$J_1(\overline{y}\cdot\overline{z}\cdot J_{01}(x\cdot\overline{x}))$
0	$\overline{1}$	0	$J_1(\overline{y}\cdot J_{01}(x\cdot\overline{x})\cdot J_{01}(z\cdot\overline{z}))$
0	$\overline{1}$	1	$J_1(\overline{y}\cdot z\cdot J_{01}(x\cdot\overline{x}))$
0	0	$\overline{1}$	$J_1(\overline{z}\cdot J_{01}(x\cdot\overline{x})\cdot J_{01}(y\cdot\overline{y}))$
0	0	0	$J_1(J_{01}(x\cdot\overline{x})\cdot J_{01}(y\cdot\overline{y})\cdot J_{01}(z\cdot\overline{z}))$
0	0	1	$J_1(z\cdot J_{01}(x\cdot\overline{x})\cdot J_{01}(y\cdot\overline{y}))$
0	1	$\overline{1}$	$J_1(y\cdot\overline{z}\cdot J_{01}(x\cdot\overline{x}))$
0	1	0	$J_1(y\cdot J_{01}(x\cdot\overline{x})\cdot J_{01}(z\cdot\overline{z}))$
0	1	1	$J_1(y\cdot z\cdot J_{01}(x\cdot\overline{x}))$
1	$\overline{1}$	$\overline{1}$	$J_1(x\cdot\overline{y}\cdot\overline{z})$
1	$\overline{1}$	0	$J_1(x\cdot\overline{y}\cdot J_{01}(z\cdot\overline{z}))$
1	$\overline{1}$	1	$J_1(x\cdot\overline{y}\cdot z)$
1	0	$\overline{1}$	$J_1(x\cdot\overline{z}\cdot J_{01}(y\cdot\overline{y}))$
1	0	0	$J_1(x\cdot J_{01}(y\cdot\overline{y})\cdot J_{01}(z\cdot\overline{z}))$
1	0	1	$J_1(x\cdot z\cdot J_{01}(y\cdot\overline{y}))$
1	1	$\overline{1}$	$J_1(x\cdot y\cdot\overline{z})$
1	1	0	$J_1(x\cdot y\cdot J_{01}(z\cdot\overline{z}))$

1 1 1 $J_1(x \cdot y \cdot z)$

Implementing this decoder circuit only needs thirty J-AND gate. If J-gate and NAND gate are used, it needs three J_0 gates, three J_1 gates, three $J_{\bar{1}}$ gates and twenty seven NAND gates and twenty seven complementary gates. The number of components is more larger.

Example 2.   Simplify a three variables decision function.

$$F = J_1(x_1) \cdot J_1(x_2) + J_1(x_1) \cdot J_1(x_3)$$
$$+ J_1(x_2) \cdot J_2(x_3) + 0 \cdot \left[J_0(x_1) \cdot J_0(x_2) \right.$$
$$\left. + J_0(x_1) \cdot J_0(x_3) + J_0(x_2) \cdot J_0(x_3) \right.$$

According to conventional method, it can not be simplified. But, the method discussed in the paper can be used to simplify further.

$$F_1 = J_1(x_1 x_2) + J_1(x_1 x_3) + J_1(x_2 x_3)$$
$$+ 0 \cdot \left[J_{01}(x_1 \overline{x_1} x_2 \overline{x_2}) + J_{01}(x_1 \overline{x_1} x_3 \overline{x_3}) \right.$$
$$\left. + J_{01}(x_2 \overline{x_2} x_3 \overline{x_3}) \right]$$
$$= J_1 \left[J_1(x_1 x_2) + J_1(x_1 x_3) + J_1(x_2 x_3) \right]$$
$$+ 0 \cdot J_1 \left[J_{01}(x_1 \overline{x_1} x_2 \overline{x_2}) \right.$$
$$\left. + J_{01}(x_1 \overline{x_1} x_3 \overline{x_3}) + J_{01}(x_2 \overline{x_2} x_3 \overline{x_3}) \right]$$

It can also be simplified as following:

$$F_2 = J_1(x_1 x_2) + J_1(x_1 x_3) + J_1(x_2 x_3)$$
$$+ 0 \cdot \left[J_{01}(x_1 \overline{x_1} x_2 \overline{x_2}) + J_{01}(x_1 \overline{x_1} x_3 \overline{x_3}) \right.$$
$$\left. + J_{01}(x_2 \overline{x_2} x_3 \overline{x_3}) \right.$$
$$= J_1(x_1 x_2 + x_1 x_3 + x_2 x_3) + 0 \cdot J_{01}(x_1 \overline{x_1} x_2 \overline{x_2}$$
$$+ x_1 \overline{x_1} x_3 \overline{x_3} + x_2 \overline{x_2} x_3 \overline{x_3})$$

We can see: F_1 and F_2 are simpler than F. It is mainly that the J-AND circuit and J-OR circuit possessed fine characteristics have been used.

IV. Discussion of circuits implementation

The implementation plan of CMOS circuits is discussed in the paper. The conventional NAND gate circuit is shown in Fig. 1. If R_1 and R_2 are symmetric, the circuits are not working in optimal state and its noise allowance is much lower than binary CMOS circuits. To raise the noise allowance, nonsymmetric resistance should be applied. Thus, the circuit will be

working in optimal state. And the value of nonsymmetric resistances varies with the number of inputs. This makes the design and manufacture very difficult. The J-NOR-gate circuit principle is shown in Fig.3.

Its characteristics are much finer than NOR gate. Only one resistance is used , and no strict requirment is set for the resistance value. J-NAND-gate shown in Fig.4 is the same as discussed above.

It is obvious that the characteristics of J-AND gate or J-OR gate are similar with J-gate, and increasing inputs will exert little influence to the circuit characteristics.

If conventional ternary gate circuit is compared to binary gate circuit, obviously, ternary is complex in structure , cost high and poor steadibility. But the J-AND gate and J-OR gate put forward in this paper with simple structure and high noise allowance, can compete with binary circuit.

The two outputs circuit can construct a complete system of its own. One of its output is J-AND operation, the other is NAND operation. It is known as J-AND complete gate. That is to say, and ternary logic function can be implemented by using J-AND complete gate.

The procedures of simplification for any ternary logic function are described as below:

1. To make use of the properties of the J-operation, we can merge the product term of the given ternary logic function, so that the least number of miniterms and input variables will produce.

Karnaugh method can also   obtain a simplified expression.

2. The rules of simplification for J-AND and J-OR operation are used to further simplify the expression obtained above. Each miniterm could be expressed as a standard form.

3. Simplification technique must be further used to replace AND operation with J-AND operation. Thus, expression with a AND operation and a OR operation can be obtained. This expression possessed the least number of gates, input variable and wires.

As for a not complex function with less variables, we can obtain a result easily through observation.
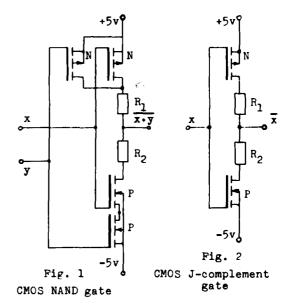
V. Conclusion

Using the method discussed above, not only the basic ternary circuit with

simpler structure, fine characteristics and low cost, can be obtained, but also the result with the least number of gates and input can be produced. Because conventional method of simplification can be used, further simplification would be made. Hence, the circuits and algebra system are advanced.

References

1. D.Michael Miller, "On the Minimization of Many-Valued Function." Proc. Ninth on MVL.

2. C.M.Allen & D.D.Givone, "A minimization technique for multiple-valued logic system". IEEE Computer Vol. C-17 pp.182-184 1968.

3. H.T.Mouftah & I.B.Jordan, " DESIGN OF TERNARY COS/MOS AND SEQUENTIAL CIRCUIT", IEEE COM. pp.281 March, 1977.

Fig. 1

CMOS NAND gate

Fig. 2

CMOS J-complement gate
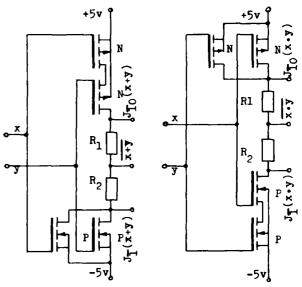
Fig. 3

J-NOR complete gate

Fig. 4

J-NAND complete gate

MICROPOWER CMOS IMPLEMENTATION OF THREE-VALUED LOGIC FUNCTIONS

Seiichi MUTA

Faculty of Engineering, Oita University
Oita-shi 870-11 Japan

ABSTRACT

A method for implementing ternary logic func-
tions with CMOS integrated circuits is proposed.
This method has the significant advantage of very
low static power consumption at any of three logic
levels comparable to that of binary CMOS circuits,
and also it needs no modification in the fabricat-
ing process of the present CMOS technology.

1. INTRODUCTION

Recently, the use of CMOS technology in the
realization of ternary (three-valued) logic func-
tions has been reported by several authors [1-6].
CMOS circuits in binary (two-valued) systems have
the important advantage of very low static power
dissipation at each logic level. But no general
method has been given to implement ternary systems
having the same advantage.

This paper proposes a method for implementing
ternary logic functions with CMOS integrated cir-
cuits which have the advantage of very low static
power consumption at any of three logic levels.
This method needs no modification in the fabricat-
ing process of the present CMOS technology.

2. PRELIMINARIES

The basic scheme of ternary circuits is shown
in Fig.1. The decoder converts each ternary input
into two binary outputs and their binary comple-
ments. The encoder combines the binary outputs of
the decoders and produces a ternary output. The
ternary logic levels are equal to 0, 1, and 2. They
correspond to the voltages $-V_{DD}$, zero potential,
and V_{DD}. The binary logic levels are 0 and 1 ($-V_{DD}$
and V_{DD}).

We define some functions [4,7].

Let $f = f(x_1, \ldots, x_n)$ be a ternary function of
n ternary variables.

Let x_i^j be a literal (binary function of a
single ternary variable) defined as

$$x_i^j = \begin{cases} 1, & \text{when } x_i \geq j \\ 0, & \text{otherwise} \end{cases}$$

where $i = 1, \ldots, n$, $j = 1, 2$.
$\overline{x_i^j}$ is the binary complement of x_i^j.
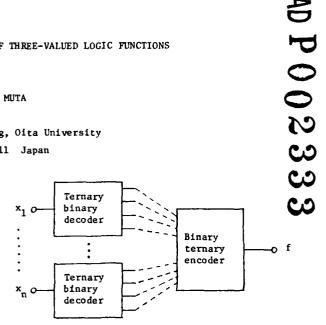
Fig.1 Scheme of ternary circuits

Then, the following relation is obtained:

$$x_i = x_i^1 + x_i^2$$

where $+$ represents arithmetic sum.

Let f^j ($j = 1, 2$) be a binary function of
2n binary variables $x_1^1, x_1^2, \ldots, x_n^1, x_n^2$ such that

$$f^j = \begin{cases} 1, & \text{when } f \geq j \\ 0, & \text{otherwise.} \end{cases}$$

Therefore the ternary function f is represented as
an arithmetic sum of two binary functions f^1 and f^2
such that

$$f = f^1 + f^2.$$

3. THE DECODER

The decoder generates four literals $x^1, x^2, \overline{x}^1$,
and $\overline{x}^2$ (for simplicity, subscripts are omitted).
The truth table is shown in Table 1. The decoder
circuit designed for CMOS 4007s is shown in Fig.2.
This circuit is a modification of a pair of CMOS
level shifter circuits and it is   constructed
so as to have the required input voltage range
(from $-V_{DD}$ to V_{DD}) and attain very low power dissi-
pation at each logic level. The resistor R and
PMOS P_9 can be removed when the decoder is fabri-
cated on a single chip, because they are used so
that internal protection circuitry of a CMOS 4007
may not make the input impedance low between $-V_{DD}$
and zero potential [8]. The dc transfer character-
istics for the circuit are shown in Fig.3. The num-
bers of transistors connected in parallel at

P_3, P_4, N_3, and N_4 give some effects on the transfer characteristics, especially increasing the number of transistors at P_4 and N_4 decreases the width of hysterisis. The supply currents through the decoder circuit are shown in Fig.4. Current i_X is caused by the above-mentioned internal protection circuitry. Fig.4 shows that the supply currents are unmeasurable for the inputs 0, 1 and 2. Therefore, the static power consumption is very low at the three logic levels.

Table 1 Decoder truth table

Input	Outputs			
x	x^1	$\overline{x}^1$	x^2	$\overline{x}^2$
2	1	0	1	0
1	1	0	0	1
0	0	1	0	1

4. THE ENCODER

A given ternary function $f(x_1,..,x_n)$ is composed of the literals $x_1^1, \overline{x}_1^1, \ldots, x_n^2, \overline{x}_n^2$ in the encoder. The encoder circuit is shown in Fig.5. P_A, P_B, and N_A are switching networks; P_A and P_B consist of PMOS's, and N_A consists of NMOS's. States of P_A, P_B, and N_A are denoted by binary variables p_A, p_B, and n_A respectively, where value 1 corresponds to closed and 0, to open.

The output y is determined by the combination of the state variables p_A, p_B, and n_A such as

$$ y = \begin{cases} 2, & \text{when } p_A \overline{p}_B \overline{n}_A = 1 \\ 1, & \text{when } \overline{p}_A p_B \overline{n}_A = 1 \\ 0, & \text{when } \overline{p}_A \overline{p}_B n_A = 1. \end{cases} $$

The networks P_A, P_B, and N_A implement binary functions as follows:

$$ p_A = f^2, \quad p_B = f^1 \overline{f^2}, \quad n_A = \overline{f^1}. $$

Fig.2 Experimental decoder circuit

$V_{DD} = 5V$, $R = 10K\Omega$.

CMOS : MC14007 , all PMOS substrates are connected to V_{DD}

P_3, P_4, N_3, N_4: two transistors are connected in parallel
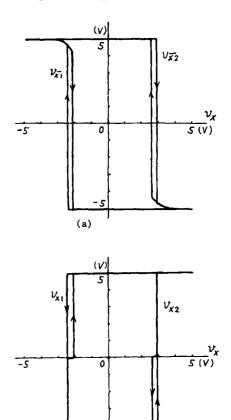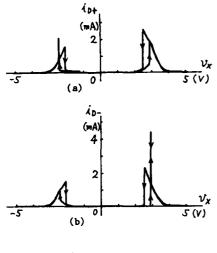
Fig.3 Dc transfer characteristics for the decoder

62
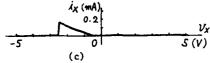
Fig.4 Supply current plots for the decoder

(a) Current i_{D+} : total current through
    V_{DD} terminals (into the decoder)
(b) Current i_{D-} : total current through
    $-V_{DD}$ terminals (out of the decoder)
(c) Current i_X : input current

By substitution, we obtain

$$y = \begin{cases} 2, & \text{when } f^2 = 1 \text{ ; } f = 2 \\ 1, & \text{when } f^1\overline{f^2} =1; \ f = 1 \\ 0, & \text{when } \overline{f^1} = 1 \text{ ; } f = 0. \end{cases}$$

Thus, we have

$$y = f(x_1,...,x_n).$$

If the number of elements of the networks is
very large, we may put a binary CMOS stage between
the decoders and the encoder [2,5,6].

5. CONCLUSION

A new method for implementing ternary logic
functions with CMOS integrated circuits has been
presented. It exhibits the significant advantage of
very low static power consumption comparable to
that of binary CMOS circuits.
We intend to extend this method to 4-valued
circuits.

REFERENCES

[1] Mouftah, H.T. and Jordan, I.B.:" Implementation
    of 3-valued logic with C.O.S.M.O.S. integrated
    circuits ", Electron. Lett., 10, 21, pp.441-442
    (Oct. 1974).
[2] Etiemble, D. and Israel, M.:" Implementation of
    ternary circuits with binary integrated cir-
    cuits ", IEEE Trans. Comput., C-26, 12,
    pp.1222-1233 (Dec. 1977).
[3] Koanantakool, H.T.:" Implementation of ternary
    identity cells using C.M.O.S. integrated cir-
    cuits ", Electron. Lett., 14, 15,pp.462-464
    (July 1978).
[4] Muta, S.:" Proposal of ternary negative gate
    circuits ", Trans. IECE Japan, J61-D, 12,
    pp.940-947 (Dec. 1978).
[5] Huertas, J.L. and Carmona, J.M.:" Low-power
    ternary C-MOS circuits ", Proc. 1979 Int. Symp.
    Multiple-Valued Logic, pp.170-174 (May 1979).
[6] McCluskey, E.J.:" A discussion of multiple-
    valued logic circuits ", Proc. 1982 Int. Symp.
    Multiple-Valued Logic, pp.200-205 (May 1982).
[7] Muta, S.:" Synthesis of ternary two-stage
    minimal negative gate networks ", Trans. IECE
    Japan, J62-D, 9, pp.592-598 (Sept. 1979).
[8] " Motorola CMOS integrated circuits ", p.5-3,
    Motorola Inc. (1978).
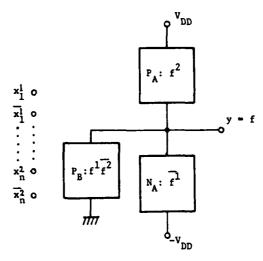
Fig.5 Encoder circuit

LOW POWER 2-OF-3-VALUED CMOS SELF-CHECKING CIRCUITS

Hu, Mou[*+‡]   Smith, K. C.[+]   Mouftah, H. T.[‡]

Abstract

Two new schemes for the implementation of self-checking binary logic systems are proposed which utilize low power 2-of-3-valued CMOS logic circuits. While 2-of-3-valued circuits are inherently ternary, only two of their three logic values are used in normal operation. The third (middle) logic value is used for self-checking and testing. To evaluate these circuits an "open-short-conducting" fault model for CMOS circuits is developed. All the single faults in these circuits are studied and classified into four types, named mid-seeking, quasi-mid-seeking, mid-rejecting, and masked. The conclusions reached for 2-of-3-valued circuits in previous papers apply to these new circuits as well. Finally a comparison between implementation schemes is made on the basis of the size of the fault set each produces.

I. Introduction

One important application of multiple-valued logic is the creation of self-checking binary logic systems using ternary circuits (1), (2), (3). In (1), a particular kind of ternary circuit with special properties was proposed. For this circuit, called a 2-of-3-valued circuit, two of the three logic values provided are used as normal binary working values, while the surplus third logic value (the middle value) is used for self-checking and testing. In (1), 2-of-3-valued combinational systems were studied. Reference (4) extended the study to 2-of-3-valued synchronous sequential systems. There it was proved that the use of 2-of-3-valued circuits can improve system reliability and simplify fault detection procedures.

* Shanghai Institute of Railway Technology, Shanghai, China

+ University of Toronto, Toronto, Ontario, Canada

‡ Queen's University, Kingston, Ontario, Canada

In this paper two new schemes to implement 2-of-3-valued circuits based on a low power CMOS technology (5) are proposed. An "open-short-conducting" fault model for these CMOS circuits is developed. All of the single faults in these circuits are studied and classified into four types: mid-seeking, quasi-mid-seeking, mid-rejecting, and masked. The conclusions made for earlier 2-of-3-valued circuits demonstrated in (1) and (4) still apply for these new 2-of-3-valued implementations. Finally, a comparison between schemes is made on the basis of the size of each fault set.

II. The 2-of-3-Valued Circuits

The first scheme for implementing 2-of-3-valued circuits was proposed in (1). These circuits are ternary logic circuits working in binary mode utilizing the two extreme logic values. The middle logic value is available for self-checking and testing.

In this section some important concepts and conclusions for 2-of-3-valued circuits will be reviewed. They are extracted from the original derivations in (1) and (4).

For compatibility with binary logic, 2-of-3-valued logic operators are defined in what follows. In each definition, variables $x, y \in Q$, where Q is the set of logic values, $Q=\{0,\frac{1}{2},1\}$. Let $N=\{0,1\}$ and $E=\{\frac{1}{2}\}$ be disjoint subsets of Q.

Definition 1

The Negation operator is defined as:

$$\bar{x}=1-x,$$

where "-" is arithmetic subtraction. The truth table of the Negation operator is given in Table 1. The circuit implementing the Negation operator will be called an Inverter.

x	$\bar{x}$
0	1
$\frac{1}{2}$	$\frac{1}{2}$
1	0

Table 1

Definition 2

The NAND operator is defined as:

$$\overline{x \cdot y} = 1 - \min(x, y),$$

where "$\min(x, y)$" implies the choice of the smaller of x and y. The truth table of the NAND is given in Table 2.

x	y	$\overline{xy}$
0	0	1
0	½	1
0	1	1
½	0	1
½	½	½
½	1	½
1	0	1
1	½	½
1	1	0

Table 2

Note from Tables 1 and 2 that when the input lies in $N = \{0, 1\}$ the truth tables revert to those of conventional binary algebra.

In 2-of-3-valued circuits, all single faults can be classified as one of four types: mid-seeking, quasi-mid-seeking, mid-rejecting, and masked. Definitions follow:

Assume that the number of inputs of a circuit G is m. With an input vector χ, and a fault f, the output of the circuit G is denoted as $G(\chi, f)$. Correspondingly the output of fault-free circuit G is denoted as $G(\chi, \phi)$.

Definition 3

A fault f is a mid-seeking fault if

$$\{\exists \chi \in N^m G(\chi, f) \in E\} \wedge \{\forall \chi \in Q^m [G(\chi, f) = G(\chi, \phi) \vee G(\chi, f) \in E]\}.$$

In other words, for a mid-seeking fault, two conditions should be met simultaneously: first that there exists at least one normal input vector, such that the output of the faulty gate is ½; and second, that for all possible input vectors, the output of the faulty gate is either correct or ½.

Definition 4

A fault f is a quasi-mid-seeking fault, if

$$\{\exists \chi \in N^m G(\chi, f) \in E\} \wedge \{\exists \chi \in Q^m [G(\chi, f) \neq G(\chi, \phi) \wedge G(\chi, f) \notin E]\}.$$

Note here, that for a quasi-mid-seeking fault, there are also two conditions. The first condition is the same as that for a mid-seeking fault. The second condition is the negation of the second condition for a mid-seeking fault.

Definition 5

A fault f is a mid-rejecting fault, if

$$\forall \chi \in Q^m \qquad G(\chi, f) \notin E.$$

In other words, for a gate having a mid-rejecting fault, the output will never be ½.

Definition 6

A fault f is a masked fault, if

$$\forall \chi \in Q^m \qquad G(\chi, f) = G(\chi, \phi).$$

In other words, for a gate having a masked fault, the output remains correct.

Now some definitions related to the self-checking concept will be provided. For these definitions, assume G is a logic circuit with m inputs and n outputs. F is the set of faults considered.

Definition 7

A logic circuit G is self-testing for F, if

$$\forall f \in F \ \exists \chi \in N^m \ G(\chi, f) \notin N^n.$$

That is, for all considered faults, there exists at least one normal input vector, such that the output vector of the circuit is abnormal.

Definition 8

A logic circuit G is fault secure for F, if

$$\forall f \in F \ \forall \chi \in N^m \{G(\chi, f) = G(\chi, \phi)\} \vee \{G(\chi, f) \notin N^n\}.$$

That is, for all faults considered, and for all normal input vectors, the output vector of the circuit is either correct or abnormal.

Definition 9

A logic circuit is totally self-checking for F, if

1) it is self-testing for F, and
2) it is fault secure for F.

As provided in (1), for mid-seeking and quasi-mid-seeking faults, the 2-of-3-valued combinational system satisfies the following theorem:

Theorem 1

For any mid-seeking and quasi-mid-seeking fault, any irredundant combinational logic network which consists of 2-of-3-valued Inverters and NAND gates is totally self-checking.

As proved in (4), for mid-seeking faults in a 2-of-3-valued synchronous sequential system the following theorem applies:

Theorem 2

If both output Z and internal state Y (or Z and next-state W) are observable, then for any mid-seeking fault, a 2-of-3-valued synchronous sequential machine is totally self-checking.

In (1), it was also proved that a 2-of-3-valued combinational system is fault secure for all masked faults, and easily testable for all mid-rejecting faults.

Similarly, a 2-of-3-valued synchronous sequential system is fault secure for all masked faults. However in a 2-of-3-valued synchronous sequential system both quasi-mid-seeking and mid-rejecting faults should be treated as hardcore and must be tested off-line.

III. Low Power CMOS 2-of-3-Valued Circuits

In this section, 2 schemes for the implementation of low power CMOS 2-of-3-valued circuits will be proposed. They are modified versions of a low power CMOS ternary family introduced earlier (5).

Scheme 1

Figure 1 is a 2-of-3-valued inverter utilizing a centre-tapped power supply. The input and output can take on values -V, 0, and +V. These correspond to logic values 0, $\frac{1}{2}$, and 1 respectively.

For proper operation it is necessary to arrange that the power supplies (-V and +V) and the thresholds of the MOSFETs (V_T) meet the following criteria:

$$V < V_T < 2V.$$

With input $\chi = -V$, P conducts, N cuts off, and the output becomes $\chi = +V$. When $\chi = 0$, both P and N cut off at which time the output takes on value 0 as supplied through R_1 and R_2. When $\chi = +V$, P cutts off, N conducts, and the output becomes $\overline{\chi} = -V$. In this circuit, 2 resistors, R_1 and R_2, are used in parallel to improve the fault detection capability.
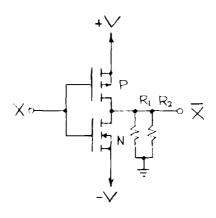
Fig. 2
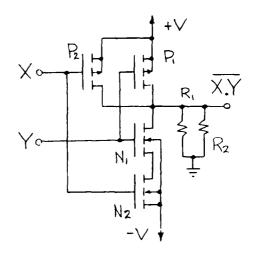
Fig. 2 shows a 2-of-3-valued NAND gate. The working principle is similar to that of the Inverter. The circuit in Fig. 2 can be augmented with additional inputs.

Scheme 2

Fig. 3 is a second 2-of-3-valued Inverter, utilizing only two power supply connections. Note that the connection of R_1 and R_2 differs from that in Fog. 1. As a result the power supply requirement is reduced from a need for matched supplies to a requirement for only a single one. However, in this circuit R_1 and R_2 have to be matched.

The required relationship between V and V_T is

$$\frac{1}{2}V < V_T < V.$$

The operation of this circuit is similar to that of the circuit of Fig. 1. The corresponding NAND gate is shown in Fig. 4

Fig. 1
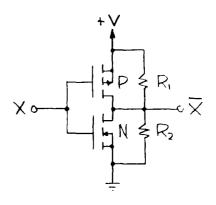
Fig. 3
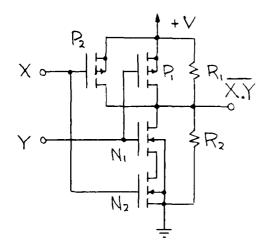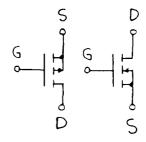
Fig. 4

Each of these gates has been built and tested using 4007, 4069, and 4011 IC's with power supply $V=1V$ and $R_1=R_2=50K$.

IV. Fault Analysis

Before we proceed to fault analysis it is necessary to establish a fault model: first of all the conventional single fault assumption is adopted, that is, that at any instant there exists at most one fault.

It is considered that a fault may be either a *resistor fault or a transistor fault. A resistor* fault results in either a resistor short condition or a resistor open condition. A transistor fault results in one of the following three conditions: source-drain (SD) shorted, SD open, or SD always conducting. Further such a fault is assumed to derive from explicit device interconnection failures. As shown by Fig. 5, if any one of the connections S, D, and G is open or if any two of them are shorted, the equivalent effect will be one of the three fault conditions: SD short, SD open, or SD always conducting*. The two conditions of SD short and SD always conducting are distinguished in view of the fact that for SD always conducting the equivalent SD resistance is not zero.

---

* It has to be noted that there could be a fourth
  condition, that is, SD always cut off. However
  in this case the current is so small that it can
  be considered as zero. Thus SD always cut off is
  considered equivalent to SD open.

Cause	Effect
D open	SD open
S open	SD open
G open	SD open or SD conducting
DG short	SD conducting
GS short	SD open
SD short	SD short

Fig. 5

With the fault model described, the fault characteristics of the new 2-of-3-valued circuits can be evaluated. The results of analysis (and experimental verification) are shown in Tables 3, 4, 5, and 6, for Inverter 1, Inverter 2, NAND 1, and NAND 2, respectively.

In these tables, the notation "S" is used to denote a resistor short or a transistor SD short, "0" is used to denote a resistor open or a transistor SD open, and "C" is used to denote a transistor which is always conducting.

x	$\bar{x}$	mid-seeking				mid-rejecting				masked	
		P_o	N_o	R_{1S}	R_{2S}	P_s	N_s	P_c	N_c	R_{10}	R_{20}
0	1	½	1	½	½	1	0	1	1	1	1
½	½	½	½	½	½	1	0	1	0	½	½
1	0	0	½	½	½	1	0	1	0	0	0

Table 3

x	$\bar{x}$	mid-seeking		mid-rejecting							
		P_o	N_o	R_{1S}	R_{2S}	P_s	N_s	P_c	N_c	R_{10}	R_{20}
0	1	½	1	1	0	1	0	1	1	1	1
½	½	½	½	1	0	1	0	1	0	0	1
1	0	0	½	1	0	1	0	1	0	0	0

Table 4

			mid-seeking						quasi-mid-seeking				mid-rejecting				masked	
X	Y	Ȳ	P_{10}	P_{20}	N_{10}	N_{20}	R_{1S}	R_{2S}	N_{1S}	N_{1C}	N_{2S}	N_{2C}	P_{1C}	P_{2C}	P_{1S}	P_{2S}	R_{10}	R_{20}
0	0	1	1	1	½	½	1	1	½	½	½	½	1	1	1	1	1	1
0	½	1	½	½	½	½	1	1	½	½	½	½	1	1	1	1	1	1
0	1	1	½	½	½	½	1	1	½	½	½	½	1	1	1	1	1	1
½	0	½	½	½	½	½	½	½	½	½	½	½	½	½	1	1	½	½
½	½	½	½	½	½	½	½	½	½	½	½	½	½	½	1	1	½	½
1	0	1	½	½	½	½	½	½	0	0	½	½	1	1	1	1	½	½
1	½	½	½	½	½	½	½	½	½	½	1	0	1	1	1	1	½	½
1	1	0	0	0	½	½	½	½	0	0	0	0	1	1	1	1	0	0

Table 5

			mid-seeking						quasi-mid-seeking				mid-rejecting					
X	Y	Ȳ	P_{10}	P_{20}	N_{10}	N_{20}	N_{1S}	N_{1C}	N_{2S}	N_{2C}	P_{1C}	P_{2C}	P_{1S}	P_{2S}	R_{1S}	R_{2S}	R_{10}	R_{20}
0	0	1	1	1	½	½	1	1	1	1	1	1	1	1	1	1	0	1	1
0	½	1	½	½	½	½	1	1	1	1	1	1	1	1	1	1	0	1	1
0	1	1	½	½	½	½	1	1	1	1	1	1	1	1	1	1	0	1	1
½	0	1	½	½	½	½	1	1	1	1	1	1	1	1	1	0	0	1	
½	½	½	½	½	½	½	½	½	1	1	1	1	1	1	1	0	0	1	
½	1	½	½	½	½	½	0	0	½	½	1	1	1	1	1	0	0	1	
1	0	1	½	½	½	½	1	1	1	1	1	1	1	1	1	0	0	1	
1	½	½	½	½	½	½	½	½	0	0	1	1	1	1	1	0	0	1	
1	1	0	0	0	½	½	0	0	0	0	1	1	1	1	1	0	0	0	

Table 6

Furthermore, following Definitions 3, 4, 5, and 6, all single faults have been classified into one of the four types.

As mentioned in Section II, for mid-seeking faults, both 2-of-3-valued combinational systems and 2-of-3-valued synchronous sequential systems are totally self-checking. Accordingly our goal is to arrange to make the set of mid-seeking faults as large as possible.

For quasi-mid-seeking faults, only the 2-of-3-valued combinational system is totally self-checking. In a 2-of-3-valued synchronous sequential system, quasi-mid-seeking faults must be treated as hard-core. On the other hand, in both the combinational system and synchronous sequential system mid-rejecting faults must be tested off-line. Thus quasi-mid-seeking faults are preferable to mid-rejecting faults.

Since masked faults do not produce errors immediately at the system output, they are undetectable by conventional means. They can however be detected using other techniques utilizing power supply current measurement. Since the accumulation of masked faults will ultimately produce errors, they must be analysed carefully.

In the inverter shown in Fig. 1, there are two masked faults, R_{10} and R_{20}. If only one of them has occurred, the system continues to operate correctly. However if both R_{10} and R_{20} occur, and input x=½ is applied, then output x becomes floating (indeterminate) instead of ½. When the circuit is under test, this constitutes an error. Furthermore this will cause the circuit to lose its self-testing ability. In other words, if a third fault occurs, even a mid-seeking fault, the system is not self-testing.

For the NAND gate shown in Fig. 2, the situation is similar. However the inverter shown in Fig. 3, and the NAND gate shown in Fig. 4, have no masked faults.

V. Scheme Comparison

To provide a meaningful comparison of the two schemes for self-checking circuits, some quantitative measures appropriate to 2-of-3-valued circuits are required.

Such has been the attempt by Lu in (6), in which some measures for self-checking circuits are defined. However they are not suited for 2-of-3-valued circuits. Thus a different approach based on the size of the four fault sets has been adopted. For each fault set, a self-checking factor C is assigned. For mid-seeking faults, the self-checking factor C_S takes value 1 since the circuit is totally self-checking. Likewise, for mid-rejecting faults, the self-checking factor C_r will be 0 since the circuit is not self-checking. For quasi-mid-seeking faults, the self-checking factor C_q takes on a value smaller than 1, yet larger than 0.

For masked faults, the value of the self-checking factor C_m varies from a negative value (see below) to a positive value less than 1 depending on the situation. If the system is used in an environment in which periodic off-line test is impossible and the mission time is quite short, masked faults are preferred to mid-rejecting faults and quasi-mid-seeking faults, since they do not cause an error at the output. Thus, $C_m > C_q$. On the other hand, if periodic off-line test is possible, and the mission time is quite long then mid-rejecting and quasi-mid-seeking faults are preferable, because masked faults are more difficult to test even in an off-line manner and because the accumulation of masked faults will ultimately cause errors. In this situation, $C_m < C_r$, requiring that it takes on a negative value.

Now let us compare the two schemes presented in Section III.

For the case in which the periodic off-line test is impossible and the mission time is quite short, the following values are assigned: $C_S=1$, $C_q=0.6$, $C_m=0.3$, and $C_r=0$. Note that the absolute values are somewhat arbitrary.

The self-checking factor for the entire circuit is: $$C = C_S \cdot \frac{N_S}{N} + C_q \cdot \frac{N_q}{N} + C_r \cdot \frac{N_r}{N} + C_m \cdot \frac{N_m}{N},$$ where N is the total number, N_S is the mid-seeking fault number, N_q is the quasi-mid-seeking fault number, N_r is the mid-rejecting fault number and N_m is the masked fault number.

For Inverter 1, N=10, $N_S=4$, $N_q=0$, $N_r=4$, and N_m

Thus, $C = 1 \times \frac{4}{10} + 0 + 0 + 0.6 \times \frac{2}{10} = 0.52$.

For Inverter 2, N=10, $N_S=2$, $N_q=0$, $N_r=8$, and N_m

Thus, $C = 1 \times \frac{2}{10} + 0 + 0 + 0 = 0.2$.

For NAND 1, N=16, $N_S=6$, $N_q=4$, $N_r=4$, and N_m

Thus, $C = 1 \times \frac{6}{16} + 0.3 \times \frac{4}{16} + 0 + 0.6 \times \frac{2}{16} = 0.52$.

For NAND 2, N=16, N_s=4, N_q=4, N_r=8, and N_m=0.

$$\text{Thus, } C=1\times\frac{4}{16}+0.3\times\frac{4}{16}+0+0=0.325$$

Obviously on this basis, scheme 1 is better than scheme 2 when off-line test is impossible and mission time is quite short.

However, if periodic off-line test is possible and the mission time is quite long, the following values can be assigned: C_s=1, C_q=0.3, C_r=0, and C_m=-0.3 in which case: For Inverter 1,

$$C=1\times\frac{4}{10}+0+0+(-0.3)\times\frac{2}{10}=0.34$$

For Inverter 2,

$$C=1\times\frac{2}{10}+0+0+0=0.2.$$

For the NAND 1,

$$C=1\times\frac{6}{10}+0.3\times\frac{4}{16}-0.3\times\frac{2}{16}=0.41.$$

For the NAND 2,

$$C=1\times\frac{4}{16}+0.3\times\frac{4}{16}+0+0=0.325.$$

Thus even for this situation, scheme 1 is still better than scheme 2. However it should be noted that scheme 2 requires only a single power supply. As well scheme 2 has no masked faults. Thus in these respects scheme 2 may be preferred to scheme 1.

II. Conclusions

In this paper, the important concepts of 2-of-3-valued circuits are reviewed and two new schemes for implementing 2-of-3-valued circuits are proposed. An "open-short-conducting" fault model is developed. Fault analysis is persued using this model. Finally a quantitative measure based on the size of each fault set is proposed. Two new schemes are compared using this method.

References

1) Mou Hu and K.C. Smith, "On the Use of CMOS Ternary Gates to Realize a Self-Checking Binary Logic System". Proceedings of the Eleventh International Symposium on Multiple-Valued Logic. pp. 212-217, May 1981.

2) K.C. Smith, "The Prospects for Multivalued Logic: A Technology and Application View", IEEE Trans. on Computers, Vol. C-30, No. 9, pp. 619-634, Sept. 1981.

3) F.F. Sellers, Jr., Mu-Yue Hsiao, and L.W. Bearnson, "Error Detecting Logic for Digital Computers", McGraw-Hill Book Company, pp. 143-145, 1968.

4) Mou Hu and K.C. Smith, "A New Type of Self-Checking Synchronous Sequential Machine Based on 2-of-3-Valued Logic Circuits". Proceedings of 12th International Symposium on Multiple-Valued Logic, pp. 139-145, May 1982.

5) H.T. Mouftah and K.C. Smith, "Injected Voltage Low Power CMOS for Three-Valued Logic", submitted for publication.

6) D.J. Lu, "Quantitative Measures and Figure of Merit for Self-Checking Circuits", CRC Technical Report No. 81-8, Stanford University Aug. 1981.

Session 2B
Threshold Logic

PREVIOUS PAGE
IS BLANK

71

SYNTHESIS OF MULTIVALUED LOGIC CIRCUITS
USING HYPERPLANES

Tatsuki Watanabe and Masayuki Matsumoto

Department of Electrical Engineering, Toyo University
Kawagoe-Shi, Japan

Abstract

This paper presents a method for synthesizing multivalued logic functions composed of hyperplanes which can be represented by multivalued multithreshold functions. In this method, multivalued logic functions can be synthesized using the operators algebraic + , - and · as well as logical max and min and no operators to yield binary values only is used. The theorem for such operations is presented. Simplified representations of functions can easily be obtained utilizing the expressions of hyperplanes. During the synthesizing process the various techniques of usual algebra and geometry are used when required. As for the circuit implementation, the "one operation amplifier per one function" method is presented.

I. Introduction

The various methods for designing multivalued logic functions in which the linear summation function of multithreshold elements is utilized have been studied by many authors [1]-[10], [12],[13]. Most of them make extensive use of the operators such as literal, inhibit or inverter which yield binary values only to establish closed intervals in the multivalued logic space. M. Davio and J.P. Deschamps [8] employed the truncated differences to realize "slopes" for values assumed by a function. Some authors [7],[12],[13] utilized hyperplanes for separation of a function, but hyperplanes are differently used in this paper. The method presented in this paper is to synthesize multivalued logic functions in which hyperplanes are not used for separation, but used as the elements of which the expressions of the functions are composed. Algebraic + , - and · as well as max and min are used as the operators. This method has a capability to yield simplified expressions of functions since the expressions of hyperplanes can be used for identifying slopes of values assumed by a function as well as for realizing the expression of the function. The techniques of usual algebra and geometry can be utilized when necessary. As for the circuit implementation, the "one operation amplifier per one function method" is also presented.

II. Definitions and Notations

In this paper, a switching algebra which has the following characteristics is employed.
1) Variables:
Variables x, y, $\cdots$ can assume at any instant of time one out of R integer values from the set $Q = \{0, 1, \cdots, R-1\}$.
2) Logical Operations:

logical sum $\quad x \vee y = \max(x, y)$
logical product $\quad x \wedge y = \min(x, y)$

3) R-valued (R-1)-threshold function $g_{R-1}^{R}(e)$ is defined as:

$$g_{R-1}^{R}(e) = r_j, \quad (t_j \leq e < t_{j+1},$$
$$j = 0, 1, \cdots, R-1) \qquad (1)$$

Where e = real-valued variable called exitation represented as (2) below,

$$r_j \in Q, \quad r_j < r_{j+1},$$
$$t_j = (r_{j-1} + r_j)/2, \quad (j = 1, \cdots, R-1);$$
$$t_0 < r_0; \quad t_R > r_{R-1}.$$

$$e = \sum_{i=0}^{n-1} w_i x_i + w_n \qquad (2)$$

where

x_i = i-th input variable

w_i, w_n = integer valued constant (either negative or positive).

Definition 1: As e expressed by (2) represents a hyperplane in the n+1 dimensional Euclidian space provided that the variables x_i are of usual real numbers, we define, in this paper, $g_{R-1}^{R}(e)$ expressed by (1) as a hyperplane (HP) in (n+1)-dimensional space Q^{n+1} where $g_{R-1}^{R}(e)$ takes values on the n+1st coordinate axis.

The functional model of a threshold element is shown in Fig. 1. In this paper, we employ the two functions of threshold elements, algebraic summation and threshold detection. The exitation e in Fig. 1 is represented by (2). Where the algebraic operators appear together with the logical operators the order of the operations for them is to be specified by parentheses as required. Furthermore, the function $P_k = g_{R-1}^{R}(e)$ is simply represented by the algebraic expression in parentheses with subscript t like

$$P_k = \left(\sum_{i=0}^{n-1} w_i x_i + w_n \right)_t \qquad (3)$$

Since such HP's expressed as (3) are of course variables in the space Q, they can be used as elements for the MV operations mentioned above. The following theorem holds.

Theorem 1: Let x, y and z be multivalued variables. The following equations hold.

$$\{x + (y \wedge z)\}_t = (x + y)_t \wedge (x + z)_t \qquad (1A)$$

72

$$\{x + (y \lor z)\}_t = (x + y)_t \lor (y + z)_t \qquad (1B)$$

$$\{x - (y \land z)\}_t = (x - y)_t \lor (x - z)_t \qquad (1C)$$

$$\{x - (y \lor z)\}_t = (x - y)_t \land (x - z)_t \qquad (1D)$$

$$\{(y \land z) - x\}_t = (y - x)_t \land (z - x)_t \qquad (1E)$$

$$\{(y \lor z) - x\}_t = (y - x)_t \lor (z - x)_t \qquad (1F)$$

Proof: Though only the proof for (1A) is shown below, the others can also be proven in the same manner.

If $y \le z$, then

$$\{x + (y \land z)\}_t = (x + y)_t. \qquad (4)$$

On the other hand,

$$(x + y)_t \le (x + z)_t$$

$$\therefore (x + y)_t \land (x + z)_t = (x + y)_t$$

$$= \{x + (y \land z)\}_t.$$

For $y \ge z$, the proof can also be made in the same manner as above. Furthermore, we can confirm those equations in Theorem 1 by the method of truth tables as in the case of Boolean algebra.

III. Representations of HP's and Their Intersections

The n-dimensional R-valued product space of n input variables $x_0, x_1, \cdots, x_{n-1}$ with values from Q is denoted as Q^n. For easy expressions, orthogonal axes are assumed for the coordinate systems throughout this paper. Let $V = (x_0, x_1, \cdots, x_{n-1})$ be any point, "vertex", in Q^n and $f(V)$ a function which assumes values on the n+1st axis y. (3) represents a HP in the space Q^{n+1} whose coordinates are $x_0, x_1, \cdots, x_{n-1}$ and y. Generally, given n+1 values assigned to each of n+1 vertices in Q^n, i.e., each of the vertices of a n-dimensional "simplex" (in terminology of mathematics [15]), it is immediate to determine a hyperplane (HP) P_k in Q^{n+1}. Some simplices in Q^n $n = 1, 2, 3$, are shown in Fig. 2.

Let the HP determined by the assigned values to a simplex S_k be denoted as P_k and the corresponding planar region included in P_k P'_k. As examples, S_0 and S_1 in Q^2 and the corresponding planar regions P'_0 and P'_1 in Q^{2+1} are shown in Fig. 3. In this paper, if a simplex is rectangular equilateral concerning a certain vertex, we call it a rectangular equilateral simplex (RELS).

The coefficients w_i, $i = 0, 1, \cdots, n$, in the expression (3) can be determined solving the simultaneous linear equations whose unknowns are $w_0, w_1, \cdots, w_n$. When the simplex by which a HP is to be determined is a RELS, the procedure to determine the expression (3) can be simplified as each of the coefficients w_i, $i = 0, \cdots, n-1$, can be immediately obtained as the slope of the HP along the corresponding lateral edge of the RELS.

Example 1: A 3-dimentional RELS $\overline{V_0 V_1 V_2 V_3}$ with the values to be taken by a HP in Q^{3+1} is shown in Fig. 4. From these values, w_i, $i = 0, 1, 2$, in (3) which represent the slopes of the HP along the respective axes x_i are immediately obtained as 3, 2 and -1 respectively. Thus, P is represented as:

$$P = (3x_0 + 2x_1 - x_2 + w_3)_t \qquad (5)$$

As the HP assumes value 0 at the vertex (1,1,1), we can get -4 for w_3.

Suppose two HP's P_0 and P_1 in Q^{n+1} having different slopes. Naturally, they intersect each other by a n-dimensional boundary simplex.

Example 2: The planar regions P'_0 and P'_1 shown in Fig. 3 whose extensions are the planes P_0 and P_1 respectively connect by a line segment. In this case, we can say that P'_0 and P'_1 are on the logical product $P_0 \land P_1$.

IV. Covers of Functions by HP's

A n-variable R-valued switching function $f(x_0, \cdots, x_{n-1})$ can be represented by a R-valued truth table. As an example, a truth table in coordinate representation for a two-variable four-valued function is given in Fig. 5.

Definition 2: When a function f has an expression composed of m HP's $P_0, P_1, \cdots, P_{m-1}$ as its elements, we call the set $\{P_0, P_1, \cdots, P_{m-1}\}$ a cover of the function f.

The following theorem holds.

Theorem 2: At least one cover always exists for a multivalued function f specified by any truth table.

Proof: It is well known that the switching algebra which includes the operators literal, max and min is functionally complete. Since we can realize an equivalent function to the operator literal by a logical product of HP's with steep slopes and the algebra in this paper employs the operators max and min, the subject has been proven.

Here, we introduce the following definition:

Definition 3: Let the space Q^n be divided into a number of n-dimensional simplices S_i without any gap and overlapping among them and also, let all the vertices to which the values taken by a function $f(x_0, x_1, \cdots, x_{n-1})$ are assigned be included as the vertices of some of such S_i. In this case, the set of S_i in Q^n constitute a "complex". In this paper, the set $\{P_i\}$ which consists of all the HP's determined by the values assigned to the vertices of such S_i is called a cover complex (CCOM) of the function f.

As far as our experiences are concerned, it seems to be true that we can synthesize a function using only the HP's within a CCOM $\{P_i\}$ of the function f. Such a function f may have a logical sum of products (or a logical product of sums) form in which the expressions of the HP's are used as the elements.

Example 3: In Fig. 6, the RELS's, i.e., the rectangular equilateral triangles with the respective names of the HP's P_{ij} to be determined by the values assigned to their vertices are shown and these RELS's form a CCOM for the function f.

V. Design Procedure

In this section, a procedure for synthesizing functions which satisfy the given truth tables using HP's as the elements is presented. Only integer slopes are used for HP's. The procedure follows the steps given below.

1) Given an incompletely specified truth table, assign suitable values to the DON'T CARE vertices of the RELS's each having at least one vertex of

assigned value 0 taking the following procedure for each of the DON'T CARE vertices. Pick up one of such DON'T CARE vertices as above. Suppose a line determined by one of the pairs of the assigned values along one of the axes on which the DON'T CARE vertex is located. If the vertices of the said pair of values are adjacent to the DON'T CARE vertex and the line takes an integer value at the DON'T CARE vertex, such value is adopted for a candidate for the value to be assigned to the DON'T CARE vertex. If there is no such line, the value 0 which is the same as one of the assigned values to the said RELS, may be assigned to the DON'T CARE vertex. Let a vertex in Q^n be denoted as $\langle x_0, x_1, \cdots, x_{n-1}\rangle$.

Example 4: In the truth table given in Fig.5, three lines determined by the pair of the values assigned to the vertices $(1,1),(2,1)$; $(3,0),(3,2)$; $(1,1),(1,2)$ indicated in bold lines in Fig. 7 are found to have the integer slopes which may provide the DON'T CARE vertices $(0,1)$, $(3,1)$ and $(1,0)$ around the vertices $(2,0)$ or $(0,2)$ with the values to be assigned. Notice that there are two such values for the vertex $(3,1)$ as indicated in Fig. 7. Each of these values are used in the following steps.

2) Produce all the RELS's in Q^n such that each of them has at least one vertex with the assigned value 0 (or the minimal value in the truth table). The HP's determined by those RELS's are the candidates for the elements to compose the product terms in the representation of f, because each product term of f need take the value 0 at those vertices to which the value 0 is assigned.

Example 5: In Fig.7(a) and (b), the rectangular equilateral triangls in which the names of the planes to be determined are indicated are the RELS's which has at least one vertex with the assigned value 0.

3) Determine each expression of the HP's scheduled in step 2. The simplification is automatically made at this stage because the same expression is given for the same HP even if the RELS's are different and consider that this procedure corresponds to the transfer cover selection algorithm [11] or that of the direct cover method [14]. Produce the truth table including the rows of those HP's as well as the function f. If HP's which take value 0 for only a part of the vertices with the assigned value 0 to f are included in the HP's produced in step 2, using those HP's produce all the logical products which takes value 0 for all the vertices with the assigned value 0 to f. To the truth table, add the rows for such product terms and the columns c, l and g which are to indicate the numbers of the vertices having the same values as f, less values than f and greater values than f respectively.

Example 6: The truth table which includes the rows of the planes obtained in Example 5 and the nine product terms produced by taking all the combinations of the HP's with the assigned value 0 at the vertex $(2,0)$ and those with the assigned value 0 at the vertex $(0,2)$ are shown in Table I. Such product terms can easily be produced as follows:

$$(P_0 \lor P_1 \lor Q_0) \land (P_5 \lor P_6 \lor Q_3) =$$

$$(P_0 \land P_5) \lor (P_0 \land P_6) \lor \cdots\cdots \lor (Q_0 \land Q_3)$$

The columns c, l and g are also added.

4) Checking the entries in the column c, find the HP's or the product terms which include the maximum number of the vertices having the same values as f. If there exist more than one such HP's or products, the following steps are to be taken for each of them. At this stage, the simplicity of expressions is also to be taken into consideration. thus, we select one of such HP's or products and hereafter, we call them preliminary covers (PCOV's) of f.

Example 7: According to the above procedure, the product $P_1 \land P_6$ in Table I is selected as the PCOV.

5) Produce a new row named as f_0 in the truth table such that its values for the respective vertices are the same as those of f exept that the values of the PCOV are assigned to the DON'T CARE vertices of f. Produce all the RELS's for f_0 such that at least one vertex of each of them has the different value from that of the PCOV.

Example 8: Referring to the PCOV obtained in Example 7, the row for f_0 having the different values from those of the PCOV at the vertices $(0,0)$ and $(3,3)$ is added to Table I. The truth tables for f_0 are reproduced in coodinate representation in Fig. 8(a) and (b) and those vertices with the different values from those of the PCOV are encircled. With respect to such vertices, all the rectangular equilateral triangles with the names of the corresponding HP's in them are shown in Fig. 8.

6) Determine the expressions for the HP's obtained in step 5 and add their rows to Table I. Fill the entries of c, l and g for the new rows. At this stage, at least one CCOM of the function f composed of the new HP's and those included in the PCOV has already been established. As stated in section IV, we can usually ("always" from our experiences) produce at least one expression of the function f composed of the HP's included in the CCOM. For convenience, we define the following two types for the vertices where f_0 has the different values from the PCOV.

Type 1: The vertices where f_0 has larger values than those of the PCOV.

Type 2: The vertices where f_0 has less values than those of the PCOV.

Refering to the entries for c, l and g, select one of the HP's having the same values as f_0 for the type 1 vertices and less values for the other vertices that have the assigned values and one of the HP's having the same values as f_0 for the type 2 vertices and larger values for the other vertices that have the assigned values. In this case the simplicity of the expressions is to be taken into account. If there is no such HP's, produce, using the new rows, the logical products or sums which have the same characteristics as the HP's mentioned above. For this, any rows in Table I can be used as well.

Make the logical sum of the PCOV and the selected HP's or the products produced for the type 1 vertices and further, produce the logical product of such sum with the selected HP's or the products produced for the type 2 vertices so that we can get the expression for f_0.

Example 9: Continuing Example 8, we begin with the vertex $(0,0)$ where f_0 has a larger value than the PCOV. Referring to the values in the entries for c, l and g, we can take Q_5 or produce the

74

product $P_7 \wedge P_8$ for the vertex $(0,0)$ and Q_6 or $P_9 \vee P_{10}$ for the vertex $(3,3)$. Because of simplicity, we select here the HP's instead of the products. In order to provide the vertex $(0,0)$ with the value 1, we make the logical sum of the PCOV and Q_5 which has the value 1 at $(0,0)$. For the vertex $(3,3)$ where f_0 has a less value than the PCOV, we produce the product of the sum obtained above with Q_6 according to the procedure. We obtain the following expression:

$$f_0 = \{(P_1 \wedge P_6) \vee Q_5\} \wedge Q_6 = (P_1 \wedge P_6 \wedge Q_6) \vee Q_5$$

$$= \{(x_0 + 2x_1 - 2)_t \wedge (2x_0 + x_1 - 2)_t \wedge$$

$$(13 - 2x_0 - 2x_1)_t\} \vee (1 - x_0 - x_1)_t \qquad (6)$$

7) Since it is not proven yet that we can always produce an expression for f from such a CCOM as stated above, we have to say that we can use the method suggested in the proof of Theorem 2 or repeat the steps 5 and 6 using the expression obtained in step 6 above as the new PCOV to conclude the process.

8) Simplify the obtained expression applying Theorem 1 if any.

9) If there remain the other PCOV's obtained in step 4, take the same steps $(5) - (8)$ for them so that we can get the other expressions for f.

Example 10: Applying (1A) in Theorem 1 to the expression of f_0 obtained in Example 9,

$$P_1 \wedge P_6 = (x_0 + 2x_1 - 2)_t \wedge (2x_0 + x_1 - 2)_t$$

$$= [\{(x_0 + 2x_1) \wedge (2x_0 + x_1)\} - 2]_t$$

$$= \{x_0 + x_1 - 2 + (x_0 \wedge x_1)\}_t.$$

Thus, we get the following expression:

$$f_0 = [\{x_0 + x_1 - 2 + (x_0 \wedge x_1)\}_t \wedge$$

$$\{1 + 2(6 - x_0 - x_1)\}_t] \vee (1 - x_0 - x_1)_t. \qquad (7)$$

VI. Functions with Several Variables

As an example, let us take the four-valued full adder circuit. Its block diagram and the truth table in coordinate representation are shown in Fig. 9 and Fig. 10 respectively. One of the 3-dimensional RELS's, i. e., a rectangular equilateral tetrahedron having $(3,1,1)$ as the right angle vertex is shown in Fig. 11. The HP's determined in Q^{3+1} by such RELS's are reduced into very few kinds. Through the steps described in Section V, we get the following representation which is considered to be the simplest.

$$f(x_0, x_1, x_2) = (S - 4)_t \vee [(S)_t \wedge \{3(4 - S)\}_t] \qquad (8)$$

where

$$S = x_0 + x_1 + x_2.$$

VII. Circuit Implementation

The circuit which represents a function can be implemented as voltage mode or current mode circuits. For current mode circuits, the usual type of I^2L circuits [3] may be used. As for voltage mode

circuits we can use the "one operation amplifier per one function" method as presented below.

Applying Theorem 1, a function composed of HP's can be converted into a form of subtraction of one of two expressions having no algebraic – from the other so that we can realize the function using only one operation amplifier. As an example, the expression of (8) can be converted as:

$$f = (S - 4)_t \vee \{(S)_t \wedge (12 - 3S)_t\}$$

$$= \{[(S - 4 + 4 + 3S) \vee \{(S + 4 + 3S) \wedge$$

$$(12 - 3S + 4 + 3S)\}] - (4 + 3S)\}_t$$

$$= \{[(4S) \vee \{(4S + 4) \wedge 16\}] - (4 + 3S)\}_t \qquad (9)$$

For simplicity of demonstration, assuming S as one variable which can take one of values $0 - 7$, we can get the circuit in Fig. 12 for eq.(9) employing the principles of the usual addition or subtraction circuits with operation amplifiers. In Fig. 12, the feed-back circuit with r_{12} and r_{13} is to compensate the deviations caused by the circuits for $\wedge$ and $\vee$ operations realized by the transistors named as T_0 and T_1 and those T_3 and T_4 respectively. This circuit can produce such exact outputs as to need no level-regeneration circuit if the inputs are correct. Notice that the outmost parentheses with sub. t of (9) is deleted in Fig. 12.

VIII. Conclusion

The developed method to implement multivalued switching functions allows comparatively simple synthesizing procedure including simplification of expressions as stated in 3) of section V. Though only the sum of products form has been considered, those of product of sums form can be treated in the similar manner. It is considered that the circuits obtained through the procedure described in this paper may have fine transient characteristics for fast operations since the functions are composed of "slopes".

References
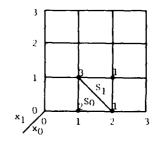
[1] T.T. Dao, E.J. McCluskey and L.K. Russel, "Multivalued Integrated Injection Logic," IEEE Trans. Comput., vol. C-26, pp. 1233-1241, Dec. 1977.

[2] O. Ishizuka, "On Multivalued Multithreshold Networks Composed of Conventional Threshold Elements," IEEE Trans. Comput. vol. C-26, pp. 1251-1257, Dec. 1977.

[3] E.J. McCluskey, "Logic Design of Multi-Valued I^2L Logic Circuits," Proc. 8th Symp. Multiple-Valued Logic, May 1978, pp. 14-22.

[4] K.W. Current and D.A. Mow, "Four-Valued Threshold Logic Full Adder Circuit Implementation," Proc. 8th Symp. Multiple-Valued Logic, May 1978, pp. 95-100.

[5] O. Ishizuka, "Synthesis of Multivalued Multithreshold Networks for Applying I^2L Circuits," Proc. 9th Symp. Multiple-Valued Logic, May

[6] E.J. McCluskey, "Logic Desighn of Multi-Input Quad I^2L Circuits," Proc. 9th Symp. Multiple-Valued Logic, May 1979, pp. 121-127.

[7] C. Moraga, "Extension of Multiple-Valued Threshold Logic," Proc. 9th Symp. Multiple-Valued Logic, May 1979, pp. 232-240.

[8] M. Davio and J.P. Deschamps, "Synthesis of Discrete Functions Using I^2L Technology," IEEE Trans. Comput., vol. C-30, Sept. 1981, pp.653-661.

[9] A. Druzeta, Z.G. Vranesic and A.S. Sedra, "Applications of Multi-Threshold Elements in the Realization of Many Valued Logic Networks," IEEE Trans. Comput., vol. C-23, Nov. 1974, pp. 1194-1198.

[10] K.C. Smith, "The Prospects for Multivalued Logic: A Technology and Applications View," IEEE Trans. Comput., vol. C-30, Sept. 1981, pp.619-634.

[11] Z.G. Vranesic, E.S. Lee and K.C. Smith, "A Many-Valued Algebra for Switching Systems," IEEE Trans. Comput., vol. C-19, Oct. 1970, pp. 964-971.

[12] T. Kitahashi, H. Nomura, Y. Tezuka and Y. Kasahara, "Characterizing Parameters of Ternary Logical Functions and their Applications to the Threshold Logical Functions" (in Japanese), Trans. IECE, vol. 52-C, No.10, pp. 641-648, Oct. 1969.

[13] H. Nomura, "Learning of the Multi-Threshold Function and its Application to the Synthesis of Logical Functions" (in Japanese), Trans. IECE, vol. 55-D, No.3, pp.194-201, March 1972.

[14] G. Pomper and J.R. Armstrong, "Representation of Multivalued Functions Using the Direct Cover Method," IEEE Trans. Comput. vol. C-30, Sept. 1981, pp. 674-679.

[15] K. Yano, "Mathematical Small Dictionary" (in Japanese), Kyoritsu Publication Co., Oct. 1968.

Table 1. The Truth Table for the Function in Q^{2+1}

		x_0: 0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3	c	l	g
		x_1: 0 0 0 0	1 1 1 1	2 2 2 2	3 3 3 3			
		f: 1 - 0 1	- 1 2 -	0 2 - 3	- 3 - 1			
P_0, P_2	$(2x_1)_t$	0 0 ⓪ 0	2 2 ② 2	3 3 3 ③	3 ③ 3 3			
P_1, P_3, Q_1, Q_2	$(x_0+2x_1-2)_t$	0 0 ⓪ ①	0 ① ② 3	2 3 3 ③	3 ③ 3 3			
P_4	$(x_0+x_1-2)_t$	0 0 ⓪ ①	0 0 1 2	⓪ 1 2 ③	1 2 3 3			
P_5	$(2x_0)_t$	0 2 3 3	0 2 3 3	⓪ 2 3 ③	0 2 3 3			
P_6, Q_4	$(2x_0+x_1-2)_t$	0 2 3 3	0 ① 3 3	⓪ 2 3 ③	1 ③ 3 3			
Q_0	$(x_1)_t$	0 0 ⓪ 0	1 ① 1 1	2 ② 2 2	3 ③ 3 3			
Q_3	$(x_0)_t$	0 1 2 3	0 ① ② 3	⓪ 1 2 ③	0 1 2 3			
$P_0 \wedge P_5$		0 0 ⓪ 0	0 2 ② 2	⓪ ② 3 ③	0 2 3 3	5		
$P_0 \wedge P_6$		0 0 ⓪ 0	0 ① ② 2	⓪ ② 3 ③	1 ③ 3 3	7	2	1
$P_0 \wedge Q_3$		0 0 ⓪ 0	0 ① ② 2	⓪ 1 2 ③	0 1 2 3	5		
$P_1 \wedge P_5$		0 0 ⓪ ①	0 ① ② 3	⓪ ② 3 ③	0 2 3 3	7	2	1
$P_1 \wedge Q_3$		0 0 ⓪ ①	0 ① ② 3	⓪ 1 2 ③	0 1 2 3	6		
$P_1 \wedge P_6$	PCOV	0 0 ⓪ ①	0 ① ② 3	⓪ ② 3 ③	1 ③ 3 3	8	1	1
$Q_0 \wedge P_5$		0 0 ⓪ 0	0 ① 1 1	⓪ ② 2 2	0 2 3 3	4		
$Q_0 \wedge P_6$		0 0 ⓪ 0	0 ① 1 1	⓪ ② 2 2	1 ③ 3 3	5		
$Q_0 \wedge Q_3$		0 0 ⓪ 0	0 ① 1 1	⓪ 1 2 2	0 1 2 3	3		
P_4		0 0 ⓪ ①	0 0 1 2	⓪ 1 2 ③	1 2 3 3	4		
f_0		① 0 ⓪ ①	0 ① ② 3	⓪ ② 3 ③	① ③ ③ ①			
P_7	$(x_0-x_1+1)_t$	① ① 2 3 3	0 ① ② 3	⓪ 0 1 2	0 0 0 ①	5	3	2
P_8	$(-x_0+x_1+1)_t$	① 0 ⓪ 0	2 ① 0 0	3 ② 1 0	3 ③ 2 ①	6	3	1
P_9	$(-2x_0+7)_t$	3 3 3 ①	3 3 3 1	3 3 3 1	3 ③ 3 ①	3	1	6
P_{10}	$(-2x_1+7)_t$	3 3 3 3	3 3 3 3	3 3 3 ③	1 1 1 ①	2	1	7
Q_5	$(-x_0-x_1+1)_t$	① 0 ⓪ 0	0 0 0 0	⓪ 0 0 0	0 0 0 0	3	7	0
Q_6	$(-2x_0-2x_1+13)_t$	3 3 3 3	3 3 3 3	3 3 3 ③	3 ③ 3 ①	3	0	7

Fig. 1. Functional model of a threshold element.

(a) Simplices S_0, S_1.

$F(x_0, x_1)$

(b) Planar region P_0', P_1'.
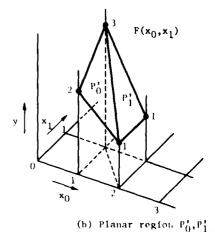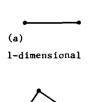
Fig. 3. Examples of simplices and planar regions.

(a)
1-dimensional

(b)
2-dimensional

(c)
3-dimensional

Fig. 2. Simplices.
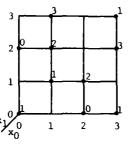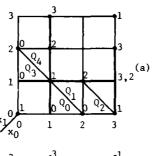
Fig. 5. A truth table in coordinate representation.

Fig. 6. A CCOM composed of RELS's.
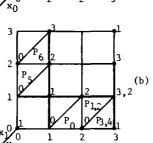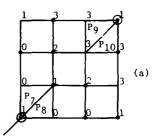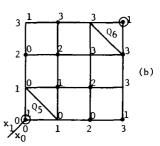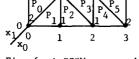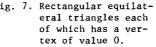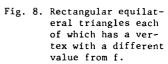
Fig. 7. Rectangular equilateral triangles each of which has a vertex of value 0.

Fig. 8. Rectangular equilateral triangles each of which has a vertex with a different value from f.
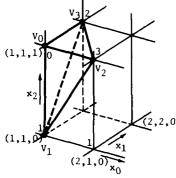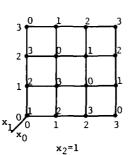
Fig. 4. A 3-dimensional RELS.

$x_2=1$

$x_2=0$

Fig. 9. Block diagram of 4-valued full adder.

Fig. 11. A RELS in Q^3.

Fig. 10. Truth table for the sum in the 4-valued full adder.

$r_0=R/2$, $r_1=R/9$, $r_3=r_4=r_5=r_7=r_9=R$

$r_6=R/6$, $r_8=R/3$, $R=18K\Omega$

Fig. 12. The circuit to realize the function f composed of HP's.

77

P-VALUED INPUT, Q-VALUED OUTPUT THRESHOLD LOGIC
AND
ITS APPLICATION TO THE SYNTHESIS OF P-VALUED LOGICAL NETWORKS

by Takahiro HAGA* and Teruo FUKUMURA**

* Education Center for Information Processing, Nagoya University
** Faculty of Engineering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya-shi, JAPAN 464

Abstract

In this paper, we describe an application of the p-valued input, q-valued output threshold elements $(2 \leq q \leq p, 3 \leq p)$, namely (p,q)-logical elements, to the synthesis of the p-valued logical networks. The idea of the (p,q)-logic stems from *the considerations that on the threshold logic the extension of the input value to the many-valued one is quite easy, while a similar extension concerning outputs is very difficult.* It is shown that under some restricted situation, the optimum value q^* of q can be determined to construct the minimum cost p-valued network using the (p,q)-threshold elements.

1. Introduction

The threshold elements will be useful for the realization of the many-valued logic, because the inputs of the threshold elements can be very easily extended to the analog-valued case and hence to the many-valued case. But the extension of the outputs of the threshold elements to the many-valued case is difficult because the number of necessary threshold *monotonically increases.*

In this paper, we propose use of the p-valued input, q-valued output threshold elements (called (p,q)-threshold elements) to construct the p-valued logical networks, where $2 \leq q \leq p$, $3 \leq p$ and the q output-values for each element are variously selected among the p values. Some p-valued logical elements are, of course, necessary at least as an output element of the p-valued logical network. Min-, max-element, or an analog-adder will be the candidate for such a p-valued logical element.

An optimum value q^* can be determined to *realize the minimum cost p-valued exclusive OR by* using the (p,q)-adic·min·max network, although the situation is rather restricted.

As a result of consideration of the (p,q)-logical completeness, the condition of completeness, which is almost equivalent to the q-valued logical completeness, is found. This result is applicable to solve what elements must be prepared to realize arbitrarily given p-valued functions by using as few kind of elements, such as polypheck[1], etc., as possible.

2. Some Properties of the p-Valued Threshold Logic[2]

In the p-valued threshold logic, one p-valued function may be a threshold function or not, depending on the p input values $\alpha_1 < \cdots < \alpha_p$, which correspond physically to the voltages, etc. By this property, whenever the physical values representing the logical values are changed, p-valued threshold network realizing a given p-valued function must be, in general, redesigned.

One method to solve such a problem is to use only the universal threshold elements which are threshold elements for any $\{\alpha_i\}$. Among the universal threshold elements, especially, the elements named p-adic elements are important.

2.1 Selection of the Logical Value

In the p-valued logical function $f = f(x_1, \cdots, x_n)$, each variable is assumed to take one of the values belonging to the set of $\{\alpha_1, \cdots, \alpha_p\}$, and also its output values are assumed to be a value in the same $\{\alpha_1, \cdots, \alpha_p\}$, where $\alpha_1 < \cdots < \alpha_p$. An n-argument p-valued function takes its values on p^n verticies $\langle x_1, \cdots, x_n \rangle$ and these verticies are properly numbered by ρ.

[Definition 1] Let a set of p logical values be denoted by L_p. L_p satisfying (1) is called symmetrical logical value and represented by S_p.

$$\alpha_1 + \alpha_p = \alpha_2 + \alpha_{p-1} = \cdots = 0 \tag{1}$$

When $\alpha_{i+1} - \alpha_i$ is equal to each other for any i in S_p, the set of such logical values is represented by D_p.

[Definition 2] A p-valued function $f(x_1, \cdots, x_n)$ is called a p-valued threshold function with respect to a set of p logical values, $L_p = \{\alpha_1, \cdots, \alpha_p\}$, when there exists a weight vector $\vec{a} = (a_1, \cdots, a_n; t_1, \cdots, t_{p-1})$ satisfying (2).

$$\left. \begin{array}{l} f(\rho) = \alpha_p \iff w(\rho) > t_{p-1} \\ f(\rho) = \alpha_{p-1} \iff t_{p-1} > w(\rho) > t_{p-2} \\ \cdots \\ f(\rho) = \alpha_1 \iff t_1 > w(\rho) \end{array} \right\} \tag{2}$$

78

where $w(\rho)=a_1\cdot x_1(\rho)+\cdots+a_n\cdot x_n(\rho)$, $x_i(\rho)\in L_p$, $t_1<\cdots<t_{p-1}$.
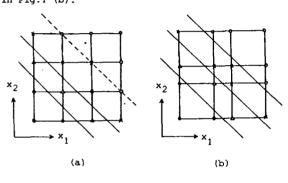
When two sets of p logical values $L_p=\{\alpha_i\}$ and $L_p'=\{\alpha_i'\}$ have a relation $\alpha_i'=c\cdot\alpha_i+d$ ($i=1,\cdots,p$; $c>0$ and d are constants), a threshold element for L_p is also so for L_p' and vice versa, therefore L_p and L_p' play the same role in the threshold logic.

[Definition 3] Define the NOT by (3).

$$\overline{\alpha}_i = \alpha_{p-i+1} \tag{3}$$

When p=2, (3) means the usual NOT of 2-valued logic. NOT of (3) corresponds to a threshold element for any L_p, and (3) is equivalent to $\overline{x}=-x$ for S_p which is easily realized.

Only S_p is considered in the following. In general, whether a given function is a threshold function or not depends on S_p. For example, the 4-valued function in Fig.1 (a) is not a threshold function for D_p but is a threshold function for S_p in Fig.1 (b).

(a)　　　　　　(b)

$$\text{output}\begin{cases} \circ\ \alpha_4, & \triangle\ \alpha_2 \\ \times\ \alpha_3, & \bullet\ \alpha_1 \end{cases}$$

Fig.1 Linear separability depending
the selection of the logical value.

2.2 p-adic Functions

The properties described in 2.1 give a problem when a given function is realized by the threshold elements. That is, by the change of the logical values the redesigns of networks might be necessary. One method solving this problem is to restrict the elements to the special elements which are threshold elements for any S_p.

[Definition 4] A function which is a threshold function for any S_p is called a universal threshold function. And, a network with only the universal elements is called a universal network.
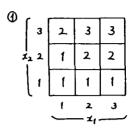
Among the universal elements, the next p-adic element is especially important.
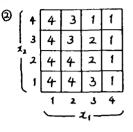
[Definition 5] Associate a p-adic number $\rho=(i_n-1,\cdots,i_1-1)_p$ with a vertex $(\alpha_{i_1},\cdots,\alpha_{i_n})$. The function $f(x_1,\cdots,x_n)$ is called a (representative) p-adic function when $f(\rho_1)\geq f(\rho_2)$ for any pair $\rho_1>\rho_2$. The families of the representative p-adic function,

that is, functions obtained from the representative function by the negation of variable and/or function, are also called p-adic functions.

For example, the functions in Table-1 are both p-adic functions.

Table-1 Examples of p-adic functions.

①

x_2		x_1	
3	2	3	3
2	1	2	2
1	1	1	1
	1	2	3

②

x_2		x_1		
4	4	3	1	1
3	4	3	2	1
2	4	4	2	1
1	4	4	3	1
	1	2	3	4

[Property 1] A p-adic function is a universal function.

(Proof) It is sufficient to show that each representative of p-adic functions is universal. And this is known easily by using a weight vector of $a_n\gg\cdots\gg a_1\geq 0$ (Q.E.D.).

The Property 1 gurantees that a network with only the p-adic elements has the universality.

3. (p,q)-adic Functions

As mentioned in 1., a p-valued threshold element needs (p-1) thresholds, and the physical realization of p-valued threshold element becomes radically difficult with the increase of p. Therefore, we consider (p,q)-adic functions with q output values among p values ($2\leq q\leq p$). The (p,q)-adic function is also regarded as an output degenerated p-adic function in 2.2. The smaller q is, the easier the physical realization of (p,q)-adic element ((q-1) thresholds suffice) is.

When p-valued networks are constructed with these (p,q)-adic elements, of course, some p-valued elements are necessary at least as an output element of the network. For such p-valued elements, analog-adder, min-, max-element, etc., are considered to be preferable, because they are easily realized and have the universal properties.

4. A Synthesis Method of the p-Valued Networks
using (p,q)-adic Elements

One method of the synthesis of p-valued networks utilizing the (p,q)-logic is proposed, which is called a (p,q)-adic·min·max network scheme as illustrated in Fig.2.

Given a p-valued function $f(x_1,\cdots,x_n)$ to be realized, the main problem is how to select the (p,q)-adic elements L_{1_1}, L_{12}, $\cdots$, L_{m1}, L_{m2} in Fig.2. One of selecting strategies is offered in Fig.3, and the strategies of (1),(2) ①, etc., always yield the sufficient (p,q)-adic elements to realize the arbitrarily given $f(x_1,\cdots,x_n)$.

Furthermore, by the strategies of (2) ②, etc., more efficient networks could be sometimes obtained. In general, to construct the networks of

79

Fig.2 (p,q)-adic·min·max network scheme.

L_{ij} is (p,q)-adic element.

(1) q=2 (2) q=3

Fig.3 The selection strategies of the (p,q)-adic elements L_{ij} in Fig.2.
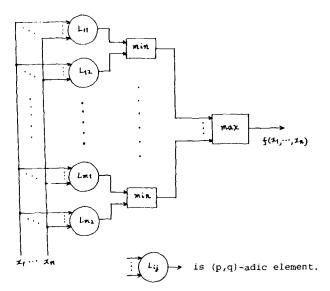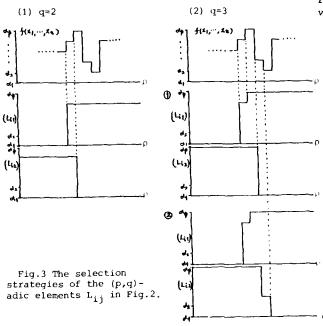
smaller m in Fig.2, it is important to reorder the variable number so that the characteristic vector of $f(x_1,\cdots,x_n)$ satisfies the relation $b_{11} \leq \cdots b_{n1}$ $(b_{i1} = \sum_{\rho} x_i(\rho) \cdot f(\rho)$, $i=1,\cdots,n)$[2], where $\rho = (i_n - 1, \cdots, i_1 - 1)_p$ of Definition 5.

5. Examples of Optimizing the Value q

Consider the p-valued exclusive OR function (4) to be realized by the scheme of Fig.2, which is important as the p-valued adder.

$$f(x_1,\cdots,x_n) = MOD[(l(x_1)-1)+\cdots +(l(x_n)-1),p]+1, \quad (4)$$

where

$$l(x) = i \quad (when \ x = \alpha_i). \quad (5)$$

For that function, the optimum q=q* will be determined in the following, under the next assumptions.

[Assumption]
(1) The realizing cost of each (p,q)-adic element is equally a(q).
(2) The realizing cost of each min-element (of 2 inputs) is equally b, where b>0.
(3) The realizing cost of max-element (of m inputs) is c·(m-1), where c>0 and b≧c in usual.

Now, two cases of a(q) are investigated,

$$\begin{cases} a(q) = a \cdot q^k & (a>0, \ k \geq 1) \quad (6-1) \\ a(q) = a^q & (a>1) \quad (6-2) \end{cases}$$

[Definition 6] Under the above assumptions, let C(q) be the total cost for constructing a network realizing a given function $f(x_1,\cdots,x_n)$. And, the value q which minimize C(q) $(2 \leq q \leq p)$ is denotated by q*, that is, $C(q^*) = \min_{2 \leq q \leq p} C(q)$.

For the function of (4), $m = p^n/(q-1)$. Therefore,

$$C(q) \cong \begin{cases} 2a \cdot q^k \cdot \frac{p^n}{q-1} + b \cdot \frac{p^n}{q-1} + c \cdot \left(\frac{p^n}{q-1} - 1\right) \\ \qquad (a(q) = a \cdot q^k) \quad (7-1) \\ 2 \cdot a^q \cdot \frac{p^n}{q-1} + b \cdot \frac{p^n}{q-1} + c \cdot \left(\frac{p^n}{q-1} - 1\right) \\ \qquad (a(q) = a^q) \quad (7-2) \end{cases}$$

$$\frac{dC(q)}{dq} \cong \begin{cases} \frac{2a \cdot (k-1) \cdot q^k - 2a \cdot k \cdot q^{k-1} - (b+c)}{(q-1)^2} \cdot p^n \\ \qquad (a(q) = a \cdot q^k) \quad (8-1) \\ \frac{2 \cdot \ln a \cdot q \cdot a^q - 2 \cdot (1 + \ln a) \cdot a^q - (b+c)}{(q-1)^2} \cdot p^n \\ \qquad (a(q) = a^q) \quad (8-2) \end{cases}$$

Table-2 Some concrete examples of q*.

(Continued)

(The case of a=2, b=c=1.)

	$a(q) = a \cdot q^k$		$a(q) = a^q$
	k=1	k≧2	
q*	p	2	3

80

	$a(\xi)=a\cdot\xi^k$			$a(\xi)=a^\xi$
	$k=1$	$k=2$	$k\geqq3$	
q^*	p	3	2	3

(The case of
a=2, b=c=4.)

As an example, some concrete values of q* are calculated in Table-2. In another special case of $a(q)=a\cdot q^2$ and $b=c$, $q^*=1+\sqrt{1+(b/a)}$ and b/a vs. q* is illustrated in Fig.4.
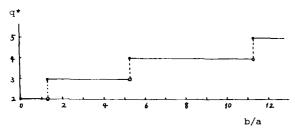
Fig.4 b/a vs. q* in a special case.

6. (p,q)-Logical Completeness

In this chapter, (p,q)-logical completeness is discussed in general, where $2\leqq q\leqq p$, $3\leqq p$ and the q output values are variously selected among the p values. As mentioned previously, some p-valued logical elements are necessary to realize p-valued logical networks.

Now, in this section, the discussion is restricted only to the *logical point of view*, so the physical values corresponding to the logical values need not be considered. Let a set L of p logical input values be

$$L = \{1,2,\cdots,p\}. \tag{9}$$

Also, let a set J of q logical output values be

$$J = \{i_1,i_2,\cdots,i_q\}, \tag{10}$$

where $1\leqq i_1<i_2<\cdots<i_q\leqq p$ $(2\leqq q\leqq p)$.

[Definition 7] Let [F] represent the set of all (p,q)-functions which are *realized by only using* the (p,q)-functions in F, where F is a set of (p,q)-functions. When [F] is equal to the set of all (p,q)-functions, F is said to be (p,q)-logically complete.

[Definition 8] A set F of (p,q)-functions is said to be closed when the following property is hold: If a (p,q)-function $f(x_1,\cdots,x_n)$ of the output-value set $J=\{i_1,\cdots,i_q\}$ is included in F, then any (p,q)-function $f'(x_1,\cdots,x_n)$ obtained by replacing output $i_1,\cdots,i_q$ to $i_1',\cdots,i_q'$, respectively, is also included in F, where $1\leqq i_1'\leqq\cdots\leqq i_q'\leqq p$.

In this Definition 8, it should be noticed that the equalities among $i_1',\cdots,i_q'$ are allowed and this is motivated by a characteristics in physical realization of the threshold elements, i.e., f and f' can be realized by the same weight vector.

[Definition 9] Let a q-valued function $f^*(x_1,\cdots,x_n)$ be obtained from a given (p,q)-function $f(x_1,\cdots,x_n)$ by restricting the input values to q values. That is,

$$f^*(x_1,\cdots,x_n)\sim f(x_1\in J_1,\cdots,x_n\in J_n) \tag{11}$$

where $J_j=\{i_{j_1},\cdots,i_{j_q}\}$, $1\leqq i_{j_1}<\cdots<i_{j_q}\leqq p$ (for each $j=1,\cdots,n$), and the symbol $\sim$ means that the q logical values in each input $x_j\in J_j$ are regarded as $1,\cdots,q$ respectively and the output values of f, say $1\leqq i_1<\cdots<i_q\leqq p$, are regarded as $1,\cdots,q$ respectively[+]. And let

$$|f(x_1,\cdots,x_n)|^* = \{f^*(x_1,\cdots,x_n)\,|\,x_1\in J_1,\cdots,$$
$$x_n\in J_n;\ \text{for any } J_1,\cdots,J_n\}. \tag{12}$$

[Definition 10] A (p,q)-function is said to be (j_1,j_2)-degenerate when (13) holds $(1\leqq j_1<j_2\leqq p)$,

$$f(x_{i_1}=\cdots=x_{i_t}=j_1;\ x_{i_{t+1}},\cdots,x_{i_n})$$
$$=f(x_{i_1}=\cdots=x_{i_t}=j_2;\ x_{i_{t+1}},\cdots,x_{i_n}) \tag{13}$$

(for any $\{x_{i_1},\cdots,x_{i_t}\}\subseteq\{x_1,\cdots,x_n\}$, $1\leqq t\leqq n$).

And, a set F of (p,q)-functions is called (j_1,j_2)-degenerate when (p,q)-functions in F are all (j_1,j_2)-degenerate for the same $1\leqq j_1<j_2\leqq p$. when there exists a (p,q)-function in F which is not (j_1,j_2)-degenerate (that is, F is not (j_1,j_2)-degenerate) for any $1\leqq j_1<j_2\leqq p$, F is called nondegenerate.

[Theorem 1] Assume a p-valued max-element is available as an output element of a network. Under this assumption, the necessary and sufficient condition for the closed set F of (p,q)-functions to be (p,q)-logically complete, is to hold the following conditions (1) and (2):

(1) $|F|^* = \{|f|^*\,|\,f\in F\}$ is q-valued logically complete.

(2) F is nondegenerate.

(Proof) Given in the Appendix.

The Theorem 1 shows that (p,q)-logical completeness is almost equivalent to q-valued logical completeness[3].

(Example) Examples of the polypheck-like (3,2)-logical completeness.

[+] If the output values of f are $1\leqq i_1<\cdots<i_q\leqq p$, $1\leqq q'\nleqq q$, then f* of (11) is not unique. In such a case, f* is defined as a set of all possible f* of (11).

$$F_3 = \left\{ \begin{array}{|c|c|c|c|} \hline 3 & 1 & 1 & 1 \\ \hline 2 & 2 & 1 & 1 \\ \hline 1 & 2 & 2 & 1 \\ \hline \end{array}, \begin{array}{|c|c|c|c|} \hline 3 & 1 & 1 & 1 \\ \hline 2 & 3 & 1 & 1 \\ \hline 1 & 3 & 3 & 1 \\ \hline \end{array}, \begin{array}{|c|c|c|c|} \hline 3 & 2 & 2 & 2 \\ \hline 2 & 3 & 2 & 2 \\ \hline 1 & 3 & 3 & 2 \\ \hline \end{array}, I_1, I_2, I_3 \right\}$$

$(I_1 \cdot 1, \ I_2 \cdot 2, \ I_3 \cdot 3.$ See the proof of
sufficiency in Appendix.)

Each F_i is generated from f_i by extending so that
F_i satisfies the closedness of the Definition 8,
and F_i could be said to be polypheck-like $(i=1,2,3)$,
i.e., each F_i is (3,2)-logically complete assuming
that 3-valued max-element is available. It should
be noticed that f_1 and f_2 are both (3,2)-adic
functions.

Now, it is known that there exists F which
satisfies only the condition (1) or (2) and F is
not (p,q)-logically complete.

(Example)

(1) F satisfies the condition (1) and is not (3,2)-logically complete.

$$F = \left\{ \begin{array}{|c|c|c|c|} \hline 3 & 2 & 1 & 1 \\ \hline 2 & 2 & 1 & 1 \\ \hline 1 & 2 & 2 & 2 \\ \hline \end{array}, \begin{array}{|c|c|c|c|} \hline 3 & 3 & 1 & 1 \\ \hline 2 & 3 & 1 & 1 \\ \hline 1 & 3 & 3 & 3 \\ \hline \end{array}, \begin{array}{|c|c|c|c|} \hline 3 & 3 & 2 & 2 \\ \hline 2 & 3 & 2 & 2 \\ \hline 1 & 3 & 3 & 3 \\ \hline \end{array}, I_1, I_2, I_3 \right\}$$

(2) F satisfies the condition (2) and is not (3,2)-logically complete.

$$F = \left\{ \begin{array}{|c|c|c|c|} \hline 3 & 1 & 2 & 2 \\ \hline 2 & 1 & 1 & 2 \\ \hline 1 & 1 & 1 & 2 \\ \hline \end{array}, \begin{array}{|c|c|c|c|} \hline 3 & 1 & 3 & 3 \\ \hline 2 & 1 & 1 & 3 \\ \hline 1 & 1 & 1 & 3 \\ \hline \end{array}, \begin{array}{|c|c|c|c|} \hline 3 & 2 & 3 & 3 \\ \hline 2 & 2 & 2 & 3 \\ \hline 1 & 2 & 2 & 3 \\ \hline \end{array}, I_1, I_2, I_3 \right\}$$

7. Conclusions

A synthesizing method of the p-valued logical
networks (that is, (p,q)-adic·min·max scheme) was
described by using the (p,q)-threshold elements
$(2 \cdot q \cdot p, \ 3 \cdot p)$. And, the optimum value q^* was
determined for the p-valued exclusive OR to be
realized by the minimum cost network. Loosely
speaking, the greater the relative cost of the
(p,q)-adic element to that of min-, max-element
becomes, the smaller the optimum value q^* becomes.
This is an intuitively acceptable result.

Next, the (p,q)-logical completeness was
discussed in general and the polypheck-like (3,2)-adic functions were discovered. A problem remains,
however, i.e., how to efficiently construct the p-valued logical networks by using a small set of
(p,q)-logical elements with min-, max- and/or
analog-adder element, etc., is left for further
studies.

References

(1) Tanaka etal:"Functional Completeness and
Polyphecks in 3-Valued Logic," Trans. IECE
Japan, J53-C, 2, pp.111-118 (1970).

(2) Haga etal:"A Problem in the Many-Valued
Threshold Logic and its Solution," Trans. IECE
Japan, J55-D, 8, pp.515-522 (1972).

(3) Rosenberg:"La structure des fonctions de
plusieurs variables sur un ensemble fini,"
Comptes Randus Acad. Sc. Paris, vol.260,
p.3817 (5 avril 1965), Groupe 1.

Appendix: Proof of Theorem 1

(1) Necessity:

Necessity of the condition (1);
If F is (p,q)-logically complete, a (p,q)-function
$f(x_1, \cdots, x_n)$ should exist for any given q-valued
function $g(x_1, \cdots, x_n)$, such that $f(x_1, \cdots, x_n)$ is
in [F] and

$$g(x_1, \cdots, x_n) \sim f(x_1 \in J_1, \cdots, x_n \in J_n)$$

where $J_j = \{ i_{j_1}, \cdots, i_{j_q} \}$ $(j=1, \cdots, n)$. F is closed,
therefore the network realizing f can be regarded
as realizing g, that is, $|F|^* = \{ |f|^* | f \in F \}$ must be
q-valued logically complete.

Necessity of the condition (2);
It suffices to note that the (j_1, j_2)-degeneracy is
preserved by the network synthesis.

(2) Sufficiency:

First, from the condition (1) and the
closedness of F, the following (p,q)-functions f_1,
f_2, f_3 and the (p,q)-functions whose output values
are variously replaced keeping the order in f_1, f_2,
f_3, are all included in [F] :

① $\bar{x} \sim f_1 (x \in J)$ (for some J)

② $x_1 \bullet \cdots \bullet x_m \sim f_2 (x_1 \in J_1, \cdots, x_m \in J_m)$
    (for some $J_1, \cdots, J_m$; for each m)

③ $x_1 + \cdots + x_m \sim f_3 (x_1 \in J_1, \cdots, x_m \in J_m)$
    (for some $J_1, \cdots, J_m$; for each m)

where $\bar{x}$ is q-valued NOT, and

$$x_1 \bullet \cdots \bullet x_m = \begin{cases} q & (x_1 = \cdots = x_m = q) \\ 1 & (\text{otherwise}) \end{cases}$$

$$x_1 + \cdots + x_m = \begin{cases} 1 & (x_1 = \cdots = x_m = 1) \\ q & (\text{otherwise}) \end{cases}$$

$(x_1, \cdots, x_m \in \{1, \cdots, q\})$.

Furthermore, from the closedness of F, the (p,1)-identity functions $I_1 \cdot 1, \cdots, I_p \cdot p$ are all included
in F. (It is easily known that $I_1, \cdots, I_p$ are all
constructed from F, even if F does not include I_1,
$\cdots, I_p$.)

Next, from the above (p,q)-functions and the
condition (2), the (p,q)-function f_4 and the (p,q)-functions whose output values are variously
replaced keeping the order in f_4, are all included
in [F]:

④ $f_4 (x \in L) = x^1 \begin{cases} q & (x=i) \\ 1 & (x=1, \cdots, i-1, i+1, \cdots, p) \end{cases}$

    (for each $i=1, \cdots, p$)

∴ By the condition (2), there exists in F a non-(j_1,j_2)-degenerate $f(x_1,\cdots,x_n)$ for any $j_1<j_2$. Let

$$\eta_{j_1,j_2}(x\in L)=f(x_{i_1}=\cdots=x_{i_t}=x\in L;$$
$$x_{i_{t+1}}=I_{k_{t+1}},\cdots,x_n=I_{k_n}),$$

where $t,k_{t+1},\cdots,k_n$ correspond to values such that

$$f(x_{i_1}=\cdots=x_{i_t}=j_1;\ x_{i_{t+1}}=k_{t+1},\cdots,x_{i_n}=k_n)$$
$$\neq f(x_{i_1}=\cdots=x_{i_t}=j_2;\ x_{i_{t+1}}=k_{t+1},\cdots,x_{i_n}=k_n).$$

then $\eta_{j_1,j_2}(x=j_1)\neq\eta_{j_1,j_2}(x=j_2)$. As F is closed, the output values of $\eta_{j_1,j_2}(x=j_1)$ and $\eta_{j_1,j_2}(x=j_2)$ can be arbitrarily selected keeping the order. Assume, without loss of generality, that $\eta_{j_1,j_2}(x=j_1)<\eta_{j_1,j_2}(x=j_2)$, then η' is obtained by ① f_1 such that $\eta'_{j_1,j_2}(x=j_1)>\eta'_{j_1,j_2}(x=j_2)$ for any $j_1<j_2$. From η and η', x^i is given as follows for each $i=1,\cdots,p$,

$$x^i=f_2(\eta_{1,i}(x),\cdots,\eta_{i-1,i}(x),$$
$$\eta_{i,i+1}(x),\cdots,\eta'_{i,p}(x)).$$

Where the output values of each η,η' are determined to be included in $J_1,\cdots,J_m$ ($m=p-1$) of ② f_2 respectively.
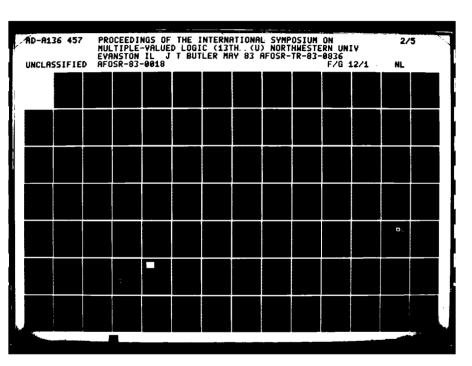
Using those functions and p-valued max-element, any (p,q)-function $f(\boldsymbol{x})=f(x_1,\cdots,x_n)$, with output-value set $\{i_1,\cdots,i_{q'}\}$, is synthesized as follows ($1\leq i_1<\cdots<i_{q'}\leq p,\ 1\leq q'\leq q$):

$$\begin{cases} f(\boldsymbol{x})=\max(h_{i_1}(\boldsymbol{x}),\cdots,h_{i_{q'}}(\boldsymbol{x})) & \text{(A-1)}\\[4pt] h_{i_j}(\boldsymbol{x})=f_3(g_{\boldsymbol{a}_1}(\boldsymbol{x}),\cdots,g_{\boldsymbol{a}_m}(\boldsymbol{x})) & \text{(A-2)}\\[2pt] \quad(\text{where }\{\boldsymbol{a}_1,\cdots,\boldsymbol{a}_m\}=\{\boldsymbol{x}\,|\,f(\boldsymbol{x})=i_j\})\\[4pt] g_{\boldsymbol{a}}(\boldsymbol{x})=f_2(x_1^{k1},\cdots,x_n^{kr}) & \text{(A-3)} \end{cases}$$

Where, (1) the output values of each x_j^{kj} in (A-3) are determined to be included in J_j of ② f_2 (for each $j=1,\cdots,n$), (2) the output values of each $g_{\boldsymbol{a}_j}(\boldsymbol{x})$ in (A-2) are determined to be included in J_j of ③ f_3 (for each $j=1,\cdots,m$), (3) the output values of each $h_{i_j}(\boldsymbol{x})$ in (A-1) are determined to take 1 and i_j (for each $j=1,\cdots,q'$).

It should be noticed that if $i_1=1$, $h_{i_1}(\boldsymbol{x})=I_1$ is sufficient (Q.E.D.).

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

GENERATION OF TERNARY MAJORITY FUNCTIONS OF FOUR OR LESS VARIABLES

Hisashi MINE   Yoshinori YAMAMOTO   Shiro FUJITA

Faculty of Engineering   Higashiyamato High   Okayama University of
Kyoto University,   School, Higashiyamato-   Science, Okayama-shi,
Kyoto-shi, 606 Japan   shi, Tokyo 189 Japan*   700 Japan

ABSTRACT

This paper proposes a method of generating ternary majority functions which have been defined in the previous paper. The method is derived from the fact that the functional form of a ternary majority function is closely related to two binary threshold functions having common weights. All the representative ternary majority functions of four or less variables are tabulated. Finally, this paper shows that the total number of ternary majority functions of four variables is 40819.

I. INTRODUCTION

Since Hanson, many contributions have been made to ternary threshold logic. Especially, a class of ternary threshold functions of three or less variables has been studied in detail, and all the functions have been shown in a table[1]-[2].

Ternary majority functions based on a ternary majority principle have been defined as an extension of binary majority functions to ternary logic[3]-[7]. It is noted that ternary majority functions differ from ternary threshold functions. Nevertheless, the functional form of a ternary majority function is closely related to two binary threshold functions[4].

Based on this property, this paper generates all the representative ternary majority functions of four or less variables by a heuristic method. Based on symmetry of the generated functions, also the total number of ternary majority functions of four or less variables is derived.

II. DEFINITIONS AND PRELIMINARY DISCUSSIONS

Let V^n be the set of all the ternary n-dimensional vectors whose components assume 0,1 or 2. A ternary logical function $f(x_1,x_2,...,x_n)$ of n variables is considered as a logical function mapping V^n into V^1. Given $x_1,x_2 \in V^1$, a ternary logical sum $\vee$, a ternary logical product $\cdot$, and complement $-$ are defined as follows:

$$x_1 \vee x_2 = \max(x_1,x_2), \quad x_1 \cdot x_2 = \min(x_1,x_2), \quad \bar{x}=2-x.$$

The notation $\cdot$ may be omitted. Any ternary n-

*Presently, Okayama University of Science.

dimensional vector $(x_1,x_2,...,x_n)$ is denoted by X.

Definition 1:
For $A=(a_1,a_2,...,a_n)$ and $B=(b_1,b_2,...,b_n)$, if $a_i \geq b_i$ $(i=1,2,...,n)$, we denote $A \geq B$. In addition to $A \geq B$, if $a_j > b_j$ for some j, we denote $A > B$.

Definition 2:
Given a ternary logical function $f(X)$, if $f(A) \geq f(B)$ for any vectors A and B satisfying $A \geq B$, f is said to be positive.

Definition 3:
Let W be a vector $(w_1,w_2,...,w_n)$. For every ternary input vector $X=(x_1,x_2,...,x_n)$, a number function $N_\theta(X)$ is defined as follows:

$$N_\theta(X) = \sum_{x_i=\theta, w_i>0} w_i - \sum_{\bar{x}_i=\theta, w_i<0} w_i, \quad \theta=1,2 \quad (1)$$

Definition 4:
A ternary logical function $f(X)$ is said to be a majority function with the structure $((w_1,w_2,..., w_n ; T_1,T_2))$ if and only if there exists a set of integer constants $w_1,w_2,...,w_n$ and $T_1,T_2 (T_1 \leq T_2)$ which satisfy the following conditions:

1) $f(X)=2 \Leftrightarrow N_2(X) \geq T_2$,

2) $f(X)=1 \Leftrightarrow \sum_{\theta=1}^{2} N_\theta(X) \geq T_1$ and $N_2(X) \leq T_2-1$, (2)

3) $f(X)=0 \Leftrightarrow \sum_{\theta=1}^{2} N_\theta(X) \leq T_1-1$.

The integer constants $w_1,w_2,...,w_n$ and T_1,T_2 are called weights and thresholds respectively.

Definition 5:
A Boolean function $g(Y)$ of n binary variables is said to be a binary threshold function with the structure $[w_1,w_2,...,w_n ; T]$ if and only if there exist a vector $W(w_1,w_2,...,w_n)$ of integer components and an integer T which satisfy

$$g(Y)=1 \Leftrightarrow WY \geq T, \qquad g(Y)=0 \Leftrightarrow WY \leq T-1, \qquad (3)$$

where Y is a binary n-dimensional input vector, and WY is an inner product of W and Y. The integer constants $w_1, w_2, \ldots, w_n$ and T are called weights and a threshold respectively.

Let $w_1, w_2, \ldots, w_n$ be weights of a ternary majority function, or a binary threshold function. If $w_1 \geq w_2 \geq \ldots \geq w_n \geq 0$ holds, the weights are said to be canonical.

Definition 6:

Given a ternary majority function $f(X)$ of n variables, let $h(X)$ be a function* derived from $f(X)$ by any combination of the following two operations(including no operation):

1. Complementation of the variables,

2. Permutation of the variables.

Then, $f(X)$ is said to be NP-equivalent to $h(X)$, and we denote $f \overset{NP}{\sim} h$.

Obviously, the binary relation $\overset{NP}{\sim}$ is an equivalence relation. Ternary majority functions are divided into equivalence classes by the NP-equivalence relation. Any ternary majority function whose weights are canonical is said to be an NP-representative of the class where the majority function belongs.

Let x_i be a ternary variable, and $\bar{x}_i$ be the complement of x_i. It is said that x_i is a positive literal and $\bar{x}_i$ is a negative literal. If a product term of some variables contains either a positive literal or a negative literal for every variable, the product term is said to be simple[8]. In particular, if the product term contains only positive literals, the product term is said to be a positive simple-product-term.

It is known that a positive ternary majority function $f(X)$ can be represented by the following logical expression:

$$f(X) = P(X) \cdot 1 \vee Q(X), \qquad (4)$$

where $P(X)$ and $Q(X)$ are logical sums of positive simple-product-terms of some variables belonging to $\{ x_1, x_2, \ldots, x_n \}$.

Let $P(X)$ be a ternary logical sum of positive simple-product-terms. Let $p(X(2))$ denote a binary logical function represented by considering ternary logical sum, ternary logical product and ternary variables in the expression of $P(X)$ as binary logical sum, binary logical product and binary variables respectively. The function $p(X(2))$ is called an associated function with the same logical expression as the one of $P(X)$, or simply an associated function with $P(X)$. For example, $p(X(2))=x_1 \vee x_2 x_3$ when $P(X)=x_1 \vee x_2 x_3$.

Property 1[4]:

Let a ternary logical function $f(X)$ be represented as Eq. (4). Let $p(X(2))$ and $q(X(2))$ be

*This function is also a majority function[3].

binary logical functions associated with $P(X)$ and $Q(X)$ respectively. The function $f(X)$ is a ternary majority function with the structure $((W ; T_1, T_2))$ if and only if $p(X(2))$ and $q(X(2))$ are binary threshold functions with the structures $[W ; T_1]$ and $[W ; T_2]$, respectively, where $W=(w_1, w_2, \ldots, w_n)$ and $T_1 \leq T_2$.

Let $f(X)$ be a ternary majority function represented by Eq. (4). Let $p(X(2))$ and $q(X(2))$ be the associated logical functions described in Prop. 1. From the latter part of Prop. 1, it is clear that, if each of the functions $p(X(2))$ and $q(X(2))$ is a representative of a class based on NP-equivalence relation[9] with respect to binary threshold functions, then the ternary majority function $f(X)$ is an NP-representative.

Consequently, in order to generate all of the NP-representatives of ternary majority functions, first, find NP-representatives $p(X(2))$ and $q(X(2))$ of binary threshold functions, which can be realized by a common weight vector. It is noted that $p(Y) \leq q(Y)$ should hold for every vector $Y \in \{0,1\}^n$ due to Prop. 1. Next, by using a unity and ternary logical expressions $P(X)$ and $Q(X)$ which associate $p(X(2))$ and $q(X(2))$ respectively, construct a logical expression by Eq. (4). A ternary logical function represented by the said logical expression gives an NP-representative among ternary majority functions.

III. BINARY THRESHOLD FUNCTIONS REALIZABLE BY COMMON WEIGHTS

In order to obtain ternary majority functions, first, we need to find binary threshold functions which have common weights as shown by Prop. 1. Muroga, Toda and Takasu, Yajima and Ibaraki, Elgot, etc.,[10] discussed on this class of binary threshold functions. It is noted that easier method than linear programming for identifying a pair of binary threshold functions with common weights are not known[10]. The exhausion of all possible simultaneaously realizable binary threshold functions has been made by Mezei[10] for the case of four variables**. In this section, we generate simultaneaously realizable binary threshold functions of four or less variables by a heuristic method.

Let $W=(w_1, w_2, \ldots, w_n)$ be an n-dimensional weight vector and Y be an n-dimensional binary input vector. An inner product WY is said to be a weighted sum of Y by W.

Given a weight vector W, let $\Delta_0, \Delta_1, \ldots$ denote weighted sums of all binary vectors belonging to $\{0,1\}^n$ by W. Let $\sim \Delta$ be a sequence of the weighted sums $\Delta_0, \Delta_1, \ldots$, which are arranged in decreasing numerical order of those values.

Definition 6:

Let $S(\Delta_i)$ represent a set of subscripts corresponding to the nonzero components of the vector Y which defines the weighted sum Δ_i. For example, $S(\Delta_i)=\{1,4,n-1\}$ when $\Delta_i = w_1+w_4+w_{n-1}$. Let $\sim \Delta$ and

**As to Mezei's work, procedures are unpublished[10].

$\sim \Delta'$ be sequences of weighted sums by weight vectors W and W' respectively. if $S(\Delta_i)=S(\Delta_i')$ for $i=0,1,2,...,2^n-1$, then the two sequences are said to be similar to each other in the logical sense.

Definition 7:

Let us consider a sequence of weighted sums by a canonical weight vector W. If any pair of weighted sums in the sequence does not coincide, then the sequence is said to be a strong sequence. This weight vector is called a universal weight vector.

It is noted that the weights of the universal weight vector W satisfy $w_1 > w_2 > w_3 > .... > w_n > 0$.

Let a strong sequence $\sim \Delta$ of the weighted sums by a universal weight vector W satisfy the following:

$$\Delta_0 > \Delta_1 > \Delta_2 > .... > \Delta_{2^n-1}(=0), \qquad (5)$$

where Δ_i is a weighted sum of an input vector Y_i. Let us find an integer T satisfying $\Delta_i \geq T > \Delta_{i+1}$. Let g(Y) be Boolean function with outputs $g(Y_j)=1$ when $\Delta_j \geq T$ and $g(Y_j)=0$ otherwise. Obviously, g(Y) is a binary threshold function with the structure [W ; T]. The function g(Y) is said to be a function generated from the sequence of Ineq. (5).

From a strong sequence, at most 2^{n+1} binary threshold functions(including constants 0 and 1) can be generated. It is noted that, if two strong sequences are similar to each other in the logical sense, the binary threshold functions generated from one strong sequence are the same as the one generated from the other. Accordingly, in order to generate binary threshold functions from strong sequences of weighted sums, it is not necessary to obtain universal weight vectors which yield similar sequences. In the sequel, the case of four-dimensional weight vectors is discussed.

All of the weighted sums by a four-dimensional weight vector $W=(w_1,w_2,...,w_n)$ are

$$0, w_1, w_2, w_3, w_4, w_1+w_2, ..., w_1+w_2+w_3+w_4.$$

Let $w_1 > w_2 > w_3 > w_4 > 0$ hold. Then the above weighted sums can be arranged in a partial order shown in Fig. 1. Let us number the weighted sums in Fig. 1 from 1 to 16 under the condition that the numbering does not contradict the above partial order. For example, the numbering in Fig. 1 is considered. Let us denote this relation by using the inequality sign as Ineq. (6).(Weighted sums are represented by only subscripts and plus sign hereafter)

1+2+3+4 > 1+2+3 > 1+2+4 > 1+3+4 > 1+2 > 1+3 > 1+4 >   (6)
2+3+4 > 1 > 2+3 > 2+4 > 3+4 > 2 > 3 > 4 > 0

In this case, $^1W=(8,4,3,2)$ is a solution of Ineq. (6) given by Sheng's method[11]. It is noted that $^1W=(8,4,3,2)$ satisfies Ineq. (6) without the equality sign. That is, ^1W is a universal weight vector. Ineq. (6) represents a strong sequence. There may
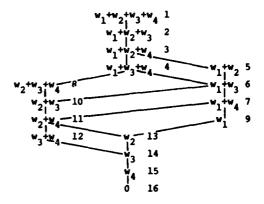
Fig. 1 Hasse diagram for a partial order.

be other vectors satisfying Ineq. (6). However, all the vectors satisfying Ineq. (6) gives the similar sequences.

All the strong sequences of weighted sums obtained from Fig. 1 and the resulting universal weight vectors are as follows:

Strong sequences of weighted sums	Universal weight vector
$\sim^1\Delta$: 1+2+3+4 > 1+2+3 > 1+2+4 > 1+3+4 > 1+2 > 1+3 > 1+4 > 2+3+4 > 1 > 2+3 > 2+4 > 3+4 > 2 > 3 > 4 > 0	$^1W=(8,4,3,2)$
$\sim^2\Delta$: 1+2+3+4 > 1+2+3 > 1+2+4 > 1+3+4 > 1+2 > 1+3 > 2+3+4 > 1+4 > 2+3 > 1 > 2+4 > 3+4 > 2 > 3 > 4 > 0	$^2W=(8,5,4,2)$
$\sim^3\Delta$: 1+2+3+4 > 1+2+3 > 1+2+4 > 1+3+4 > 1+2 > 1+3 > 2+4 > 1 > 2+3+4 > 2+3 > 2+4 > 3+4 > 2 > 3 > 4 > 0	$^3W=(10,4,3,2)$
$\sim^4\Delta$: 1+2+3+4 > 1+2+3 > 1+2+4 > 1+2 > 1+3+4 > 2+3+4 > 1+3 > 2+3 > 1+4 > 2+4 > 1 > 2 > 3+4 > 3 > 4 > 0	$^4W=(8,7,4,2)$
$\sim^5\Delta$: 1+2+3+4 > 1+2+3 > 1+2+4 > 1+3+4 > 2+3+4 > 1+2 > 1+3 > 2+3 > 1+4 > 2+4 > 3+4 > 1 > 2 > 3 > 4 > 0	$^5W=(7,6,5,3)$
$\sim^6\Delta$: 1+2+3+4 > 1+2+3 > 1+2+4 > 1+3+4 > 1+2 > 2+3+4 > 1+3 > 1+4 > 2+3 > 2+4 > 1 > 3+4 > 2 > 3 > 4 > 0	$^6W=(8,6,4,3)$*
$\sim^7\Delta$: 1+2+3+4 > 1+2+3 > 1+2+4 > 1+2 > 1+3+4 > 1+3 > 1+4 > 2+3+4 > 1 > 2+3 > 2+4 > 2 > 3+4 > 3 > 4 > 0	$^7W=(10,6,3,2)$
$\sim^8\Delta$: 1+2+3+4 > 1+2+3 > 1+2+4 > 1+3+4 > 2+3+4 > 1+2 > 1+3 > 1+4 > 2+3 > 2+4 > 3+4 > 1 > 2 > 3 > 4 > 0	$^8W=(8,6,5,4)$
$\sim^9\Delta$: 1+2+3+4 > 1+2+3 > 1+2+4 > 1+2 > 1+3+4 > 1+3 > 2+3+4 > 1+4 > 2+3 > 1 > 2+4 > 2 > 3+4 > 3 > 4 > 0	$^9W=(10,7,4,2)$
$\sim^{10}\Delta$: 1+2+3+4 > 1+2+3 > 1+2+4 > 1+3+4 > 1+2 > 1+3 > 2+3+4 > 2+3 > 1+4 > 1 > 2+4 > 3+4 > 2 > 3 > 4 > 0	$^{10}W=(10,7,6,2)$

*In Mezei's work, a weight vector (10,8,5,4) is produced. However, since this vector yields the same sequence as $\sim^6\Delta$, it is essentially identical to ^6W. Other universal weight vectors coincide with the vectors obtained by Mezei.

$\sim^{11}\Delta$: $1+2+3+4 > 1+2+3 > 1+2+4 > 1+2 > \quad ^{11}W=(10,8,4,3)$
$1+3+4 > 2+3+4 > 1+3 > 1+4 > 2+3 >$
$2+4 > 1 > 2 > 3+4 > 3 > 4 > 0$

$\sim^{12}\Delta$: $1+2+3+4 > 1+2+3 > 1+2+4 > 1+3+4 > ^{12}W=(10,8,6,3)$
$1+2 > 2+3+4 > 1+3 > 2+3 > 1+4 >$
$2+4 > 1 > 3+4 > 2 > 3 > 4 > 0$

$\sim^{13}\Delta$: $1+2+3+4 > 1+2+3 > 1+2+4 > 1+2 > \quad ^{13}W=(8,6,4,1)$
$1+3+4 > 1+3 > 2+3+4 > 2+3 > 1+4 >$
$1 > 2+4 > 2 > 3+4 > 3 > 4 > 0$

$\sim^{14}\Delta$: $1+2+3+4 > 1+2+3 > 1+2+4 > 1+2 > \quad ^{14}W=(8,4,2,1)$
$1+3+4 > 1+3 > 1+4 > 1 > 2+3+4 >$
$2+3 > 2+4 > 2 > 3+4 > 3 > 4 > 0$

Let ^1F, ^2F, ... , ^{14}F be sets of the binary threshold functions generated from the above strong sequences $\sim^1\Delta$, $\sim^2\Delta$, ... , $\sim^{14}\Delta$ respectively. Then the following theorem holds:

Theorem 1:

Let $\sim\omega$ be a sequence constructed from an arbitrary four-dimensional canonical weight vector Ω. Let G be a set of the functions generated from $\sim\omega$. Then there exists at least one set ^iF ($i \in \{1,2,..,14\}$) which satisfies the following condition:

$$G \subseteq {}^iF, \qquad (7)$$

where the notation $\subseteq$ means a usual inclusion relation between sets.

Proof* Obviously from the way of obtaining strong sequences $\sim^1\Delta$, $\sim^2\Delta$, ... , $\sim^{14}\Delta$, there exists at least one sequence which is similar in the logical sense to $\sim\omega$ among $\sim^1\Delta$, $\sim^2\Delta$, ..., $\sim^{14}\Delta$. Let one such sequence be $\sim^m\Delta$. The sequence $\sim\omega$ has no equality sign, or has at least one equality sign. The former case means that $\sim\omega$ is a strong sequence. Since $\sim\omega$ is similar in the logical sense to $\sim^m\Delta$, the theorem holds. In the latter case, we can not determine a threshold value at the place where the sign is that of equality. However, by determining threshold values at the places where inequality signs are to be found, we can generate the same binary threshold functions from $\sim\omega$ as those generated from $\sim^m\Delta$. This completes the proof. ∎

It is impossible to generate threshold functions from the sequences of weighted sums by all of the four-dimensional weight vectors in order to find common weight vectors, since infinitely many vectors exist. However, according to Theorem 1, we can generate all the desired functions by using only strong sequences $\sim^1\Delta$, ... , $\sim^{14}\Delta$. Next, let us consider a case of three variables. In this case, w_4 can be set to 0 for a weight vector (w_1, w_2, w_3, w_4) where $w_1 > w_2 > w_3 > w_4$. Only the following two strong sequences of weighted sums are considered as the ones which satisfy the above condition:

$\sim$a: $1+2+3+4=1+2+3 > 1+2+4=1+2 > 1+3+4=1+3 > 2+3+4 >$
$2+3 > 1+4=1 > 2+4=2 > 3+4=3 > 4 > 0$

---

*This theorem is considered to be essentially identical to Theorem 8.1.2.2 of [10]. However, since the way of discussions differ from those of [10], we give a proof.

$\sim$b: $1+2+3+4=1+2+3 > 1+2+4=1+2 > 1+3+4=1+3 > 1+4 >$
$2+3+4=2+3 > 2+4=2 > 3+4=3 > 4 > 0$

The sequences $\sim$a and $\sim$b are similar in the logical sense to $\sim^{13}\Delta$ and $\sim^{14}\Delta$ respectively. Thus $\sim$a and $\sim$b are omitted. The case of two or less variable functions also yield some sequences similar in the logical sense to either $\sim^{13}\Delta$ or $\sim^{14}\Delta$. Thus the sequences corresponding to two or less variable functions are omitted.

Consequently, in order to generate all of the required NP-representatives of four or less variables, it is necessary and sufficient to deal the four variable case.

IV. SYNTHESIS OF TERNARY MAJORITY FUNCTIONS

This section describes a method of synthesizing all of the NP-representatives of ternary majority functions by using the strong sequences obtained in the previous section. The method is implemented as follows:

Let $\sim^k\Delta$ be one of the strong sequences. Let all of the binary threshold functions generated from this sequence be

$$^1f, \; ^2f, \; ^3f, \; .... , \; ^mf. \qquad (8)$$

Furthermore, let thresholds of the generated binary threshold functions be $T(1)$, $T(2)$, $T(3)$,.. ,$T(m)$ respectively.

Let us consider a pair $(^if, ^jf)$ of two functions ^if and ^jf which satisfy

$$T(i) \leq T(j) \qquad (9)$$

Since ^if and ^jf are positive, these can be represented as irredundant logical expressions having no negative variables. Let ^iE and ^jE be ternary logical expressions which associate ^if and ^jf respectively (If a binary function $^if=1$, corresponding ternary function $^iE=2$. Also if $^jf=0$, $^jE=0$). Construct a logical expression

$$^iE \cdot 1 \lor {}^jE \quad \text{(1: Constant value)}. \qquad (10)$$

From the functions ^1f, ^2f, .. ,^mf, consider pairs which satisfy Ineq. (9) and construct ternary logical expressions in the above way.

Apply the above way to each of the strong sequences $\sim^1\Delta$,... , $\sim^{14}\Delta$, and construct ternary logical expressions. For example, let us choose a sequence $\sim^1\Delta$ and a universal weight vector $^1W=(8, 4,3,2)$. From this sequence, 17 different binary threshold functions are generated (including constant values 1 and 0). Among them, consider a pair of the generated functions $^if=x_1x_2 \lor x_1x_3 \lor x_1x_4$ and $^jf=x_1x_2 \lor x_1x_3x_4$. Thresholds of these functions are 10 and 12 respectively, and Ineq. (9) is satisfied. By the associate ternary logical expressions:
$^iE=x_1x_2 \lor x_1x_3 \lor x_1x_4$ and $^jE=x_1x_2 \lor x_1x_3x_4$, construct

$$(x_1x_2 \lor x_1x_3 \lor x_1x_4) \cdot 1 \lor x_1x_2 \lor x_1x_3x_4$$
$$=(x_1x_3 \lor x_1x_4) \cdot 1 \lor x_1x_2 \lor x_1x_3x_4.$$

87

A ternary logical function represented by the above logical expresion is a ternary majority function with the structure ((8,4,3,2 ; 10, 12)). Since the binary threshold functions generated in the implementation procedure are NP-representatives, each of the constructed ternary logical expressions represents an NP-representative of ternary majority functions. Then, all of the NP-representatives of ternary majority functions of four or less variables are systematically synthesized.

Ternary logical expressions which associate NP-representatives of binary threshold functions of four or less variables are shown in Table 2. They are numbered from 1 to 27(including constants 0 and 2). Hereafter, the numbers refer to these logical expressions. The structures of NP-representatives of ternary majority functions are shown in Table 3 (Constant values 0, 1 and 2 are included). A pair (a,b) in the table represents a logical expression $A \cdot 1 \vee B$ by logical expressions of No.a and B of No. b. Note that if the same functions as those obtained by the preceding sequences are synthesized from the current sequences, they are deleted from the table. Hence, the table has no duplication of functions. Generation of ternary majority functions has been implemented by an electronic computer.

Number of ternary majority functions

Finding the total number of ternary majority functions is an important problem considering the capability of ternary majority functions. This problem is, however, difficult to be solved in the case of general n variables. In this paper, an accurate number of $n (\leq 4)$ variable ternary majority functions is obtained below by considering symmetries of NP-representatives.

In the case of exactly four variables, there are 18 representatives symmetric in all variables, 61 representatives symmetric in three variables, 127 representatives symmetric in two variables, 46 representatives symmetric in each of two pairs of variables and 29 asymmetric representatives. In a similar way, check the symmetries and asymmetries of exactly three, or two variable functions. Based on the number obtained, calculate the number of all the different functions which are derived from NP-representatives by permutation and negation of variables. The result is shown in Table 1, where n variable functions include ones having dummy variables.

V. CONCLUSION

The method of synthesizing all the ternary majority functions has been discussed by means of properties on the weights of binary threshold functions. NP-representatives of ternary majority functions of four or less variables have been found, 335 in total. An accurate number of ternary majority functions of four or less variables has also been found, 40819 in the case of four variables. There remains the problem of devising an effective method of obtaining strong sequences in the case of five or more variables. Ternary regular functions have been defined by M. Mukaidono[8]. It is noted that ternary majority functions are included by the ternary majority functions. It is an interesting exercise to investigate further relationship between ternary majority functions and the ternary regular functions.

Table 1 The number of n variable ternary majority functions

n	The number of functions
1	9
2	59
3	993
4	40819

Table 2 Ternary logical expressions(denoted by only subscripts) corresponding to binary threshold NP-representatives

No.	Logical expressions	No.	Logical expressions
1	1234	14	$12 \vee 13 \vee 23 \vee 14$
2	$1 \vee 2 \vee 3 \vee 4$	15	$12 \vee 13 \vee 14 \vee 234$
3	$123 \vee 124$	16	$12 \vee 13 \vee 14$
4	$1 \vee 2 \vee 34$	17	$1 \vee 234$
5	$123 \vee 124 \vee 134 \vee 234$	18	123
6	$12 \vee 13 \vee 23 \vee 14 \vee 34 \vee 24$	19	$1 \vee 2 \vee 3$
7	$123 \vee 124 \vee 134$	20	$12 \vee 13 \vee 23$
8	$1 \vee 23 \vee 24 \vee 34$	21	$12 \vee 13$
9	$12 \vee 134 \vee 234$	22	$1 \vee 23$
10	$12 \vee 13 \vee 23 \vee 14 \vee 24$	23	12
11	$12 \vee 134$	24	$1 \vee 2$
12	$1 \vee 23 \vee 24$	25	1
13	$12 \vee 13 \vee 234$	26	Constant 0
		27	Constant 2

REFERENCES

[1] Aibara, T. and Akagi, M., "Generation of ternary threshold functions of up to three variables", (in Japanese) Trans. IECE Japan, vol. 53-C, no. 9, pp. 591-598 (Sep. 1970).

[2] Nazarala, J. and Moraga, C., "Minimal realization of ternary threshold functions", Proc. 4th ISMVL, pp. 347-359 (May 1974).

[3] Yamamoto, Y and Fujita, S., "Three-valued majority functions", (in Japanese) Trans IECE Japan, vol. J63-D, no. 6, pp. 493-500 (Jun. 1980).

[4] Mine, H. and Yamamoto, Y., "Testing and realization of three-valued majority functions", Proc. 11th ISMVL, pp. 157-162 (May 1981).

[5] Yamamoto, Y. and Fujita, S., "Multi-valued majority functions with the same logical expressions as binary threshold functions", (in Japanese) Trans. IECE Japan, vol. J64-D, no. 2, pp. 172-173 (Feb. 1981).

[6] Mine, H., Yamamoto, Y. and Fujita, S., "A method of determining the functional form of three-valued majority functions", Trans. IECE

Japan, vol.E.64, no.9, pp.604-605(Sep. 1981).

[7] Kitahashi, T., "Social decision and ternary majority functions", Proc. 12th ISMVL, pp. 159-162 (May 1982).

[8] Mukaidono, S., "Regular ternary logic functions", Proc. 13th ISMVL (May 1983).

[9] Muroga, S., Ibaraki, T. and Kitahashi, T., "Threshold logic", (in Japanese) Sangyo Tosyo Press, p. 116 (1976).

[10] Muroga, S., "Threshold logic and its applications", Wiley and Sons, pp. 215-229 (1971).

[11] Sheng, C.L., "A method for testing and realization of threshold functions", IEEE Trans., vol. EC-13, pp. 232-239 (1964).

Table 3 NP-representatives of ternary majority functions of four or less variables

Functions		w_1	w_2	w_3	w_4	;	T_1	T_2	Functions		w_1	w_2	w_3	w_4	;	T_1	T_2
27	27	8	4	3	2	;	0	0	4	16	8	4	3	2	;	4	10
27	2	8	4	3	2	;	0	1	4	21	8	4	3	2	;	4	11
27	19	8	4	3	2	;	0	3	4	11	8	4	3	2	;	4	12
27	4	8	4	3	2	;	0	4	4	7	8	4	3	2	;	4	13
27	8	8	4	3	2	;	0	5	4	3	8	4	3	2	;	4	14
27	12	8	4	3	2	;	0	6	4	18	8	4	3	2	;	4	15
27	22	8	4	3	2	;	0	7	4	1	8	4	3	2	;	4	16
27	17	8	4	3	2	;	0	8	4	26	8	4	3	2	;	4	18
27	15	8	4	3	2	;	0	9	8	8	8	4	3	2	;	5	5
27	16	8	4	3	2	;	0	10	8	12	8	4	3	2	;	5	6
27	21	8	4	3	2	;	0	11	8	22	8	4	3	2	;	5	7
27	11	8	4	3	2	;	0	12	8	17	8	4	3	2	;	5	8
27	7	8	4	3	2	;	0	13	8	15	8	4	3	2	;	5	9
27	3	8	4	3	2	;	0	14	8	16	8	4	3	2	;	5	10
27	18	8	4	3	2	;	0	15	8	21	8	4	3	2	;	5	11
27	1	8	4	3	2	;	0	16	8	11	8	4	3	2	;	5	12
27	26	8	4	3	2	;	0	18	8	7	8	4	3	2	;	5	13
2	2	8	4	3	2	;	1	1	8	3	8	4	3	2	;	5	14
2	19	8	4	3	2	;	1	3	8	18	8	4	3	2	;	5	15
2	4	8	4	3	2	;	1	4	8	1	8	4	3	2	;	5	16
2	8	8	4	3	2	;	1	5	8	26	8	4	3	2	;	5	18
2	12	8	4	3	2	;	1	6	12	12	8	4	3	2	;	6	6
2	22	8	4	3	2	;	1	7	12	22	8	4	3	2	;	6	7
2	17	8	4	3	2	;	1	8	12	17	8	4	3	2	;	6	8
2	15	8	4	3	2	;	1	9	12	15	8	4	3	2	;	6	9
2	16	8	4	3	2	;	1	10	12	16	8	4	3	2	;	6	10
2	21	8	4	3	2	;	1	11	12	21	8	4	3	2	;	6	11
2	11	8	4	3	2	;	1	12	12	11	8	4	3	2	;	6	12
2	7	8	4	3	2	;	1	13	12	7	8	4	3	2	;	6	13
2	3	8	4	3	2	;	1	14	12	3	8	4	3	2	;	6	14
2	18	8	4	3	2	;	1	15	12	18	8	4	3	2	;	6	15
2	1	8	4	3	2	;	1	16	12	1	8	4	3	2	;	6	16
2	26	8	4	3	2	;	1	18	12	26	8	4	3	2	;	6	18
19	19	8	4	3	2	;	3	3	22	22	8	4	3	2	;	7	7
19	14	8	4	3	2	;	3	4	22	17	8	4	3	2	;	7	8
19	8	8	4	3	2	;	3	5	22	15	8	4	3	2	;	7	9
19	12	8	4	3	2	;	3	6	22	16	8	4	3	2	;	7	10
19	22	8	4	3	2	;	3	7	22	21	8	4	3	2	;	7	11
19	17	8	4	3	2	;	3	8	22	11	8	4	3	2	;	7	12
19	15	8	4	3	2	;	3	9	22	7	8	4	3	2	;	7	13
19	16	8	4	3	2	;	3	10	22	3	8	4	3	2	;	7	14
19	21	8	4	3	2	;	3	11	22	18	8	4	3	2	;	7	15
19	11	8	4	3	2	;	3	12	22	1	8	4	3	2	;	7	16
19	7	8	4	3	2	;	3	13	22	26	8	4	3	2	;	7	18
19	3	8	4	3	2	;	3	14	17	17	8	4	3	2	;	8	8
19	18	8	4	3	2	;	3	15	17	15	8	4	3	2	;	8	9
19	1	8	4	3	2	;	3	16	17	16	8	4	3	2	;	8	10
19	26	8	4	3	2	;	3	18	17	21	8	4	3	2	;	8	11
4	4	8	4	3	2	;	4	4	17	11	8	4	3	2	;	8	12
4	8	8	4	3	2	;	4	5	17	7	8	4	3	2	;	8	13
4	12	8	4	3	2	;	4	6	17	3	8	4	3	2	;	8	14
4	22	8	4	3	2	;	4	7	17	18	8	4	3	2	;	8	15
4	17	8	4	3	2	;	4	8	17	1	8	4	3	2	;	8	16
4	15	8	4	3	2	;	4	9	17	26	8	4	3	2	;	8	18

Table 3 (Continued)

Left half

Functions		w₁	w₂	w₃	w₄	;	T₁	T₂
15	15	8	4	3	2	;	9	9
15	16	8	4	3	2	;	9	10
15	21	8	4	3	2	;	9	11
15	11	8	4	3	2	;	9	12
15	7	8	4	3	2	;	9	13
15	3	8	4	3	2	;	9	14
15	18	8	4	3	2	;	9	15
15	1	8	4	3	2	;	9	16
15	26	8	4	3	2	;	9	18
16	16	8	4	3	2	;	10	10
16	21	8	4	3	2	;	10	11
16	11	8	4	3	2	;	10	12
16	7	8	4	3	2	;	10	13
16	3	8	4	3	2	;	10	14
16	18	8	4	3	2	;	10	15
16	1	8	4	3	2	;	10	16
16	26	8	4	3	2	;	10	18
21	21	8	4	3	2	;	11	11
21	11	8	4	3	2	;	11	12
21	7	8	4	3	2	;	11	13
21	3	8	4	3	2	;	11	14
21	18	8	4	3	2	;	11	15
21	1	8	4	3	2	;	11	16
21	26	8	4	3	2	;	11	18
11	11	8	4	3	2	;	12	12
11	7	8	4	3	2	;	12	13
11	3	8	4	3	2	;	12	14
11	18	8	4	3	2	;	12	15
11	1	8	4	3	2	;	12	16
11	26	8	4	3	2	;	12	18
7	7	8	4	3	2	;	13	13
7	3	8	4	3	2	;	13	14
7	18	8	4	3	2	;	13	15
7	1	8	4	3	2	;	13	16
7	26	8	4	3	2	;	13	18
3	3	8	4	3	2	;	14	14
3	18	8	4	3	2	;	14	15
3	1	8	4	3	2	;	14	16
3	26	8	4	3	2	;	14	18
18	18	8	4	3	2	;	15	15
18	1	8	4	3	2	;	15	16
18	26	8	4	3	2	;	15	18
1	1	8	4	3	2	;	16	16
1	26	8	4	3	2	;	16	18
26	26	8	4	3	2	;	18	18
27	14	8	5	4	2	;	0	9
27	13	8	5	4	2	;	0	11
2	14	8	5	4	2	;	1	9
2	13	8	5	4	2	;	1	11
19	14	8	5	4	2	;	3	9
19	13	8	5	4	2	;	3	11
4	14	8	5	4	2	;	5	9
4	13	8	5	4	2	;	5	11
8	14	8	5	4	2	;	6	9
8	13	8	5	4	2	;	6	11
12	14	8	5	4	2	;	7	9
12	13	8	5	4	2	;	7	11
22	14	8	5	4	2	;	8	9
22	13	8	5	4	2	;	8	11
14	14	8	5	4	2	;	9	9
14	15	8	5	4	2	;	9	10
14	13	8	5	4	2	;	9	11
14	21	8	5	4	2	;	9	12

Right half

Functions		w₁	w₂	w₃	w₄	;	T₁	T₂
14	11	8	5	4	2	;	9	13
14	7	8	5	4	2	;	9	14
14	3	8	5	4	2	;	9	15
14	18	8	5	4	2	;	9	16
14	1	8	5	4	2	;	9	18
14	26	8	5	4	2	;	9	20
15	13	8	5	4	2	;	10	11
13	13	8	5	4	2	;	11	11
13	21	8	5	4	2	;	11	12
13	11	8	5	4	2	;	11	13
13	7	8	5	4	2	;	11	14
13	3	8	5	4	2	;	11	15
13	18	8	5	4	2	;	11	16
13	1	8	5	4	2	;	11	18
13	26	8	5	4	2	;	11	20
27	25	10	4	3	2	;	0	10
2	25	10	4	3	2	;	1	10
19	25	10	4	3	2	;	3	10
4	25	10	4	3	2	;	4	10
8	25	10	4	3	2	;	5	10
12	25	10	4	3	2	;	6	10
22	25	10	4	3	2	;	7	10
17	25	10	4	3	2	;	8	10
25	25	10	4	3	2	;	10	10
25	16	10	4	3	2	;	10	11
25	21	10	4	3	2	;	10	13
25	11	10	4	3	2	;	10	14
25	7	10	4	3	2	;	10	15
25	3	10	4	3	2	;	10	16
25	18	10	4	3	2	;	10	17
25	1	10	4	3	2	;	10	18
25	26	10	4	3	2	;	10	20
27	24	8	7	4	2	;	0	7
27	10	8	7	4	2	;	0	9
27	20	8	7	4	2	;	0	11
27	9	8	7	4	2	;	0	13
27	23	8	7	4	2	;	0	15
2	24	8	7	4	2	;	1	7
2	10	8	7	4	2	;	1	9
2	20	8	7	4	2	;	1	11
2	9	8	7	4	2	;	1	13
2	23	8	7	4	2	;	1	15
19	24	8	7	4	2	;	3	7
19	10	8	7	4	2	;	3	9
19	20	8	7	4	2	;	3	11
19	9	8	7	4	2	;	3	13
19	23	8	7	4	2	;	3	15
4	24	8	7	4	2	;	5	7
4	10	8	7	4	2	;	5	9
4	20	8	7	4	2	;	5	11
4	9	8	7	4	2	;	5	13
4	23	8	7	4	2	;	5	15
24	24	8	7	4	2	;	7	7
24	12	8	7	4	2	;	7	8
24	10	8	7	4	2	;	7	9
24	14	8	7	4	2	;	7	10
24	20	8	7	4	2	;	7	11
24	13	8	7	4	2	;	7	12
24	9	8	7	4	2	;	7	13
24	11	8	7	4	2	;	7	14
24	23	8	7	4	2	;	7	15
24	3	8	7	4	2	;	7	16

Table 3   (Continued)

Functions		w_1	w_2	w_3	w_4	;	T_1	T_2
24	18	8	7	4	2	;	7	18
24	1	8	7	4	2	;	7	20
24	26	8	7	4	2	;	7	22
12	10	8	7	4	2	;	8	9
12	20	8	7	4	2	;	8	11
12	9	8	7	4	2	;	8	13
12	23	8	7	4	2	;	8	15
10	10	8	7	4	2	;	9	9
10	14	8	7	4	2	;	9	10
10	20	8	7	4	2	;	9	11
10	13	8	7	4	2	;	9	12
10	9	8	7	4	2	;	9	13
10	11	8	7	4	2	;	9	14
10	23	8	7	4	2	;	9	15
10	3	8	7	4	2	;	9	16
10	18	8	7	4	2	;	9	18
10	1	8	7	4	2	;	9	20
10	26	8	7	4	2	;	9	22
14	20	8	7	4	2	;	10	11
14	9	8	7	4	2	;	10	13
14	23	8	7	4	2	;	10	15
20	20	8	7	4	2	;	11	11
20	13	8	7	4	2	;	11	12
20	9	8	7	4	2	;	11	13
20	11	8	7	4	2	;	11	14
20	23	8	7	4	2	;	11	15
20	3	8	7	4	2	;	11	16
20	18	8	7	4	2	;	11	18
20	1	8	7	4	2	;	11	20
20	26	8	7	4	2	;	11	22
13	9	8	7	4	2	;	12	23
13	23	8	7	4	2	;	12	15
9	9	8	7	4	2	;	13	13
9	11	8	7	4	2	;	13	14
9	23	8	7	4	2	;	13	15
9	3	8	7	4	2	;	13	16
9	18	8	7	4	2	;	13	18
9	1	8	7	4	2	;	13	20
9	26	8	7	4	2	;	13	22
11	23	8	7	4	2	;	14	15
23	23	8	7	4	2	;	15	15
23	3	8	7	4	2	;	15	16
23	18	8	7	4	2	;	15	18
23	1	8	7	4	2	;	15	20
23	26	8	7	4	2	;	15	22
27	6	7	6	5	3	;	0	8
27	5	7	6	5	3	;	0	14
2	6	7	6	5	3	;	1	8
2	5	7	6	5	3	;	1	14
19	6	7	6	5	3	;	4	8
19	5	7	6	5	3	;	4	14
4	6	7	6	5	3	;	6	8
4	5	7	6	5	3	;	6	14

Functions		w_1	w_2	w_3	w_4	;	T_1	T_2
8	6	7	6	5	3	;	7	8
8	10	7	6	5	3	;	7	9
8	20	7	6	5	3	;	7	11
8	9	7	6	5	3	;	7	13
8	5	7	6	5	3	;	7	14
6	6	7	6	5	3	;	8	8
6	10	7	6	5	3	;	8	9
6	14	7	6	5	3	;	8	10
6	20	7	6	5	3	;	8	11
6	13	7	6	5	3	;	8	12
6	9	7	6	5	3	;	8	13
6	5	7	6	5	3	;	8	14
6	7	7	6	5	3	;	8	15
6	3	7	6	5	3	;	8	16
6	18	7	6	5	3	;	8	17
6	1	7	6	5	3	;	8	19
6	26	7	6	5	3	;	8	22
10	5	7	6	5	3	;	9	14
10	7	7	6	5	3	;	9	15
14	5	7	6	5	3	;	10	14
20	5	7	6	5	3	;	11	14
20	5	7	6	5	3	;	11	15
13	5	7	6	5	3	;	12	14
9	5	7	6	5	3	;	13	14
9	7	7	6	5	3	;	13	15
5	5	7	6	5	3	;	14	14
5	7	7	6	5	3	;	14	15
5	3	7	6	5	3	;	14	16
5	18	7	6	5	3	;	14	17
5	1	7	6	5	3	;	14	19
5	26	7	6	5	3	;	14	22
10	15	8	6	4	3	;	9	11
15	9	8	6	4	3	;	11	13
24	22	10	6	3	2	;	6	9
24	17	10	6	3	2	;	6	10
24	15	10	6	3	2	;	6	11
24	16	10	6	3	2	;	6	12
24	21	10	6	3	2	;	6	13
22	23	10	6	3	2	;	9	16
17	23	10	6	3	2	;	10	16
15	23	10	6	3	2	;	11	16
16	23	10	6	3	2	;	12	16
21	23	10	6	3	2	;	13	16
6	15	8	6	5	4	;	9	12
15	5	8	6	5	4	;	12	15
22	20	10	7	6	2	;	10	13
20	21	10	7	6	2	;	13	16
24	25	8	4	2	1	;	4	8
25	23	8	4	2	1	;	8	12

Session 3A
Logic Design II

ON THE NUMBER OF LOCATIONS REQUIRED

IN THE CONTENT-ADDRESSABLE MEMORY IMPLEMENTATION

OF MULTIPLE-VALUED FUNCTIONS

Jon T. Butler

Department of Electrical Engineering
and Computer Science
Northwestern University
Evanston, IL 60201

ABSTRACT

A multiple-valued function can be realized by a binary content addressable memory (CAM), a decoder which converts multiple-valued inputs to binary addresses, and a decoder which converts the binary CAM outputs to a multiple-valued output. Of particular interest is the number of CAM locations required in a specific implementation. In this paper, an upper bound on the storage requirements is derived for m-valued n-input functions. This is compared with the storage requirements for specific functions, such as the MIN and MAX functions. Also, the average number of locations is computed for m-valued n-input functions and is shown to be somewhat more than one-half of the upper bound.

I. INTRODUCTION.

The difficulty of developing a technology capable of more than two levels of logic has prevented the production of commercially available circuits employing multiple-valued logic exclusively. Until a fast, high-radix technology emerges, we will see hybrid circuits, combining binary with some higher radix logic. An example is the INTEL 432 and 8087[1], in which a 4-valued ROM control memory is embedded in a binary-valued IC. While state-of-the-art VLSI technology can produce a 4-valued uniformly structured ROM, 4-valued random logic is still not at hand. Thus, the hybrid circuit.

Another application of the hybrid circuit is in systems where the cost of the interconnect is large. In aircraft control systems, such as SIFT[2], processors, sensors, and actuators are distributed throughout the aircraft and interconnected by busses whose weight can be a substantial fraction of the aircraft weight. Multiple-valued signalling offers promise, not only for data lines but for control lines as well. For example, Vranesic and Zaky[3] have proposed the use of multiple-valued data and control signals in a local network.

Papachristou[4] has proposed a hybrid circuit in which multiple-valued combinational logic functions are implemented with binary-valued content-addressable memories (CAM). An advantage of the CAM implementation is that it is direct, requiring the truth table of the target function. A decoder converts the multiple-valued inputs to binary-valued address lines for the CAMs, and an encoder converts the binary CAM outputs to a multiple-valued output. An interesting problem is the choice of the encoder function, as it determines the number of CAM locations required to implement a given function.

Let $L_{min}(m,n)$ be the minimum number of CAM locations required to implement a given m-valued n-input (1-output) function. Papachristou[4] has shown that

$$((m^{n+1}/2 - m^2/2 - m/2)\log_2 m$$

is an upper bound on $L_{min}(m,n)$. In this paper, an upper bound better by a factor of m is shown. Further, it is shown that the bound is firm when m is a power of 2; there exist functions with minimum storage requirements which equal the upper bound.

Also considered in this paper is the storage requirement for specific functions, the MAX, MIN, mod m addition, and mod m multiplication functions. Finally, the average number of CAM locations required for m-valued n-input functions is calculated and compared with the maximum number of CAM locations.

II. CAM IMPLEMENTATION OF MULTIPLE-VALUED LOGIC FUNCTIONS.

Consider an m-valued n-input function. Such a function can be realized as shown in Fig. 1 below. The n m-valued inputs are applied to a decoder which produces binary address lines that are applied to all CAMs. Each CAM produces a 1 or 0 depending on whether or not it contains the address applied. The collective CAM outputs form a binary codeword which is applied to the encoder whose output is the single m-valued output.

For a given m-valued n-input function, it is of interest to produce the CAM implementation of lowest cost. The cost is determined by the complexity of the decoder, the CAMs, and the encoder.
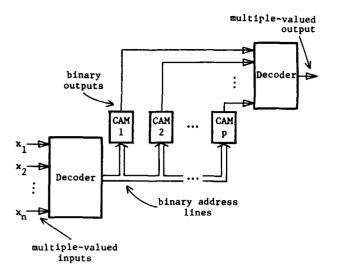
94

Figure 1. CAM Implementation of an m-valued n-input Function.

In the analysis to follow, it is assumed that, for a fixed m and n, the complexity of the decoder and encoder is fixed (or negligibly small) and that the complexity of the CAM is measured by the total number of locations required. The important design parameter then is the encoding function, since it determines the latter. To see this, consider two CAM implementations of the 3-valued function shown below in Table I. In Implementation #1, the enco-

			Implementation #1 01-0, 00-1, & 10-2		Implementation #2 00-0, 01-1, & 10-2	
x_1	x_2	q	CAM_1	CAM_2	CAM_1	CAM_2
0	0	0	0	1	0	0
0	1	0	0	1	0	0
0	2	0	0	1	0	0
1	0	0	0	1	0	0
1	1	1	0	0	0	1
1	2	0	1	0	1	0
2	0	0	0	1	0	0
2	1	2	1	0	1	0
2	2	1	0	0	0	1

Table I. Two CAM Implementations of a Three-Valued Function.

der realizes the function (01-0, 00-1, and 10-2), while the encoder of Implementation #2 realizes (00-0, 01-1, and 10-2). The output of the two CAMs is shown in Table I for each implementation. Recall that an output of 1 indicates the presence of a storage location. Thus, Implementation #1 requires 7 locations, while Implementation #2 requires 4, and so the latter is preferred. Consider now the problem of finding the minimal cost CAM

implementation of an arbitrary multiple-valued function.

Let **recurrency** w_k be the number of occurrences of logic value k in the output column of a multiple-valued function. For example, for the function of Table I, $w_0 = 5$, $w_1 = 2$, and $w_2 = 2$. The sum of the recurrencies must be the total number of entries in the truth table, and so,

$$\sum_{i=0}^{m-1} w_i = m^n ,$$

where m is the number of logic values and n is the number of inputs.

Associate with each m-valued output j a unique binary codeword which has v_j 1's. v_j is called the **weight** of j. For example, in Implementation #1, $v_0 = 1$, $v_1 = 0$, and $v_2 = 1$, while for Implementation #2, $v_0 = 0$, $v_1 = 1$, and $v_2 = 1$.

The **total CAM storage** $L(m,n)$ of a given m-valued n-input function is

$$L(m,n) = \sum_{i=0}^{m-1} w_i v_i . \qquad (1)$$

For example, for Implementations #1 and #2, we have $L(m,n)$ as 7 and 4, respectively.

III. WORST CASE NUMBER OF STORAGE LOCATIONS.

For any multiple-valued logic function, the number of storage locations needed in the CAMs is given by (1). Writing (1) to show two recurrency values w_j and w_k explicity, gives

$$L(m,n) = w_j v_j + w_k v_k + \sum_{i \neq j, k} w_i v_i .$$

Assume $v_j > v_k$. If $w_j > w_k$, then a new distribution of recurrency values obtained by interchanging the binary codewords associated with logic values j and k produces a total storage less than the original.

Therefore, given any distribution of recurrencies to codewords, a rearrangement which associates the largest recurrency with 00...0, the codeword of weight 0; the next $C(p,1)$ largest with the $C(p,1)$ codewords of weight 1; the next $C(p,2)$ largest with the $C(p,2)$ codewords of weight 2; etc. has the smallest storage requirement, where $p = \lceil \log_2 m \rceil$ is the number of bits in the codeword ($\lceil d \rceil$ is the integer equal to or just larger than d) and $C(p,i)$ is the number of combinations of p things taken i at a time. Such a distribution is called **monotone decreasing**. For example, the distribution associated with Implementation #2 ($w_0=5$ 00-0, $w_1=2$ 01-1, and $w_2=2$ 10-2) is monotone decreasing, while

95

that of Implementation #1 ($w_0 = 2$ 01-0, $w_1 = 5$ 00-1, and $w_2 = 2$ 10-2) is not. Implementation #2, therefore, has the least number of storage locations of any CAM implementation of the function of Table I. Let $L_{min}(m,n)$ be the minimum number of locations required in the CAM implementation of a given m-valued n-input function as required by the monotone decreasing recurrency distribution. For example, for the function of Table I, $L_{min}(3,2) = 4$.

Now consider all m-valued n-input functions. For each, there is a minimum CAM implementation. Of this set of implementations, we are interested in the worst case; i.e. the largest of the $L_{min}(m,n)$'s. Papachristou[4] has shown that

$$L_{min}(m,n) \leq (m^{n+1}/2 - m^2/2 - m/2)\log_2 m.$$

An improved (smaller) upper bound is derived here by showing a set of functions whose minimum storage requirements exceeds all others.

For the analysis to follow, it is convenient to consider only recurrencies which are monotone decreasing, where $w_0 \geq w_1 \geq \ldots \geq w_{m-1}$. Thus, 00...0 will encode 0, 10...0 will encode 1, 01...0 will encode 2, etc.. Note that the minimum CAM implementation of a function with the same set of recurrencies arranged in a different order will have the same cost; although the decoder function will be different, the number of CAM locations will be the same. Further, we will assume that the number of bits in the codewords $p = \lceil \log_2 m \rceil$. A **uniform recurrency distribution** is a monotone decreasing recurrency distribution in which the recurrencies are identical; i.e. $w_0 = w_1 = \ldots = w_{m-1}$. We have the following result.

Lemma 1: An m-valued n-input function realizes the largest minimum CAM storage requirement $L_{min}(m,n)$ of all m-valued n-input functions iff it has a uniform recurrency distribution.

Proof: (if) On the contrary, assume there is a function G with the largest minimum CAM storage requirement which does not have a uniform recurrency distribution. The monotone decreasing distribution of G can be converted into a uniform distribution by a sequence of steps of the form: For each pair of recurrencies (w_j, w_i) such that $w_j \geq w_i + 1$, $w_j > w_{j+1}$, and $w_{i-1} > w_i$, decrease w_j by 1 and increase w_i by 1. The minimum storage requirement never decreases at each step, and so a uniform recurrency distribution exists with a largest $L_{min}(m,n)$.

(only if) One can proceed from a uniform recurrency distribution to any monotone decreasing distribution by a sequence of steps which are the converse of the steps in the if part above. At each step, the memory size remains the same or increases. On the first step, the memory size always increases. Thus, the uniform recurrency distribution uniquely attains the maximum memory size.

Q.E.D.

For large m, Papachristou's upper bound on $L_{min}(m,n)$ is approximately $(m^{n+1}/2)\log_2 m$. This bound is improved as follows. From Lemma 1, an upper bound on the number of CAM storage locations required to implement an m-valued n-input function is the number of locations required by functions with a uniform recurrency distribution. In such a distribution, there are $m^n/m = m^{n-1}$ input tuples which map to each output logic value. p-bit codewords represent each output logic value, where $p = \lceil \log_2 m \rceil$. There are $C(p,i)$ p-bit codewords with i 1's, and each input tuple whose output value corresponds to such a codeword requires i storage locations. Thus, the total number of storage locations associated with input tuples whose output value is encoded by a codeword with i 1' is $m^{n-1}iC(p,i)$. Summing over all i yields,

$$L_{min}(m,n) \leq m^{n-1} \sum_{i=0}^{p} i\, C(p,i) . \qquad (2)$$

The righthand side of (2) is strictly larger than the lefthand side when p is not a power of 2, since there are fewer output logic values than p-bit codewords. The righthand side can be evaluated as follows. From the binomial theorem [5, p.17],

$$(x + 1)^p = \sum_{i=0}^{p} C(p,i)\, x^i . \qquad (3)$$

Differentiating the righthand side of (3) with respect to x and setting x = 1, yields the sum on the righthand side of (2). Thus,

$$L_{min}(m,n) \leq m^{n-1}(d/dx)(x + 1)^p \Big|_{x=1} = m^{n-1}p2^{p-1} .$$

Substituting for p yields,

$$L_{min}(m,n) \leq m^{n-1}\lceil \log_2 m \rceil (2^{\lceil \log_2 m \rceil}/2) . \qquad (4)$$

When m is a power of 2, $p = \log_2 m$, and

$$L(m,n) \leq (m^n/2)\log_2 m \qquad (5)$$

is a firm upper bound; it expresses exactly the maximum number of locations required in the minimal implementation of the highest cost m-valued n-input functions.

IV. **STORAGE REQUIREMENTS FOR SPECIFIC MULTIPLE-VALUED FUNCTIONS.**

In this section, we derive the storage requirements for certain multiple-valued functions and compare this with the worst case number.

(a) MAX-MIN Functions.

Let the **MAX (MIN)** function

$$G_1 = MAX(x_1, x_2, \ldots, x_n) \quad (G_2 = MIN(x_1, x_2, \ldots, x_n))$$

be an m-valued n-input function in which the output value is the maximum (minimum) of the input values. Because the monotone decreasing recurrency distributions for these functions is the same, it is sufficient to consider just the MAX function. Let $T_n(i)$ be the number of n-tuples of values 0 to m-1 in which i is the largest value. Thus, $T_n(i)$ is the number of input tuples which map to i under the MAX function. There are $(i+1)^n$ n-tuples of values 0 to i and i^n tuples of values 0 to i-1. Thus, the number of n-tuples with at least one i but no larger value is

$$T(i) = (i + 1)^n - i^n , \qquad 0 \leq i \leq m-1 .$$

The monotone decreasing distribution of recurrencies for the MAX (MIN) functions is

$$m^n - (m-1)^n \geq (m-1)^n - (m-2)^n \geq \ldots \geq 1^n - 0^n.$$

To realize the MAX function with the least amount of CAM storage, we associate the output value of largest recurrency, $m^n - (m-1)^n$, with 00...0. The output values with the next $p = C(p,1)$ largest recurrencies, $(m-1)^n - (m-2)^n$, $(m-2)^n - (m-3)^n$, ... , and $(m-C(p,1))^n - (m-C(p,1)-1)^n$ are encoded by 10...0, 01...0, ... , and 00...1, etc.. Thus, we have,

$$L_{min}(G_1) = L_{min}(G_2) = \sum_{i=0}^{p} i \sum_{j=0}^{C(p,i)-1} (m - \sum_{k=0}^{i-1} (C(p,k)-j))^n - (m - \sum_{k=0}^{i-1} C(p,k) - (j+1))^n. \quad (6)$$

After rearrangement, (6) becomes

$$L_{min}(G_1) = L_{min}(G_2) = \sum_{i=0}^{a} (m - \sum_{j=0}^{i} C(p,j))^n , \qquad (7)$$

where a is the largest integer such that

$$m \geq \sum_{j=0}^{a} C(p,j) .$$

When the number of inputs n is large, the first term of (7) dominates, and we can write,

$$L_{min}(G_1) = L_{min}(G_2) \sim (m-1)^n ,$$

where $f(n) \sim g(n)$ means $\lim f(n)/g(n) = 1$, as $n \to \infty$. Table IV, at the end of this section, shows the

values of $L_{min}(G_1)$ and $L_{min}(G_2)$ for $3 \leq m \leq 8$ and $2 \leq n \leq 5$.

(b) Modulo Addition.

Let

$$G_3 = MODSUM(x_1, x_2, \ldots, x_n)$$

be an m-valued n-input function in which the output value is the sum of the input values modulo m. All output values occur in equal number, and this function produces the uniform recurrency distribution. Thus, the CAM storage requirement for the MODSUM is the maximum over m-valued n-input functions. The storage requirement can be calculated in a straightforward manner for specific m and n. For example, for m = 3, 4, 5, 6, 7, and 8, the cost is $2 \cdot 3^{n-1}$, $4 \cdot 4^{n-1}$, $5 \cdot 5^{n-1}$, $7 \cdot 6^{n-1}$, $9 \cdot 7^{n-1}$, and $12 \cdot 8^{n-1}$, respectively. Table IV shows the costs for these six values of m and for $2 \leq n \leq 5$.

(c) Modulo Multiplication.

Let

$$G_4 = MODPROD(x_1, x_2, \ldots, x_n)$$

be an m-valued n-input function in which the output value is the product of the input values modulo m. For this function, 0 has the highest recurrency, while the relative recurrencies of other values depend on how many factors in their prime factorization are shared with m. To see this, consider multiplication modulo 4. For n = 2, we have the multiplication table of Table II.

x_2 \ x_1	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Table II. $G_4(x_1, x_2)$ Verses x_1 and x_2.

Let $w_i(n)$ be the number of occurrences of output value i in the MODPROD of n input variables. For example, $w_0(2) = 8$, $w_1(2) = 2$, $w_2(2) = 2$, and $w_3(2) = 4$. We can calculate these recurrencies recursively from

$$G_4(x_1,x_2,\ldots,x_n) = \text{MODPROD}(G_4(x_1,x_2,\ldots,x_{n-1}),x_n).$$

Consider $w_0(n)$. $G_4(x_1,x_2,\ldots,x_n)$ is 0 for all four values of $G_4(x_1,x_2,\ldots,x_{n-1})$ when $x_n = 0$, for one value (0) of $G_4(x_1,x_2,\ldots,x_{n-1})$ when $x_n = 1$, for two values (0 and 2) of $G_4(x_1,x_2,\ldots,x_{n-1})$ when $x_n = 2$, and for one value (0) of $G_4(x_1,x_2,\ldots,x_{n-1})$ when $x_n = 3$. Thus, we can write

$$w_0(n) = 4w_0(n-1) + w_1(n-1) + 2w_2(n-1) + w_3(n-1).$$

The other w_i's can be derived in a similar manner.

$$w_1(n) = w_1(n-1) + w_3(n-1),$$

$$w_2(n) = w_1(n-1) + 2w_2(n-1) + w_3(n-1),$$

and $\quad w_3(n) = w_1(n-1) + w_3(n-1).$

This set of recursion relations can be solved [5, pp. 60-73] to produce closed-form expressions,

$$w_0(n) = 4^n - n2^{n-1} - 2^n,$$

$$w_1(n) = w_3(n) = 2^{n-1},$$

and $\quad w_2(n) = n2^{n-1}.$

Table III shows the closed-form expressions for the

recurrencies of the m-valued n-input MODPROD function for $3 \leq m \leq 8$. Also shown are expressions for the total number of storage locations required. It is interesting to compare the relative storage requirements for the various values of m. When m is a prime, the number of locations required is $O((m-1)^n)$. Thus, for increasing m, when m is prime, the cost increases. However, when composite m's are included, quite a different situation exists. For example, the cost of an n-input 6-valued MODPROD function is less than that of an n-input 5-valued function!

An examination of the MODPROD function truth table for a prime m shows that the logic values 1, 2, ..., m-1 occur in equal numbers. Such a distribution is more expensive to implement than the non-uniform distribution of the MODPROD function for composite m.

Table IV, on the next page, shows that the number of storage locations required by the MODPROD function equals or exceeds the number required for the MIN and MAX functions when m, the number of logic values, is prime. However, when m is composite, the opposite is true. This difference is especially notable when n is very large. In fact, the number of CAM storage locations required for the MODPROD function becomes a vanishingly small fraction of the locations required by the MIN or MAX function as n approaches ∞, when m is composite. When m is prime, the cost of the MODPROD function is a constant (somewhat larger than 1) times the cost of the MIN or MAX function.

Number of Output Logic Values m	Expressions for Recurrencies of m-valued n-input MODPROD Functions	Expressions for CAM Storage Requirements
3	$w_0(n)=3^n-2^n$, $w_1(n)=w_2(n)=2^{n-1}$	2^n
4	$w_0(n)=4^n-n2^{n-1}-2^n$, $w_1(n)=w_3(n)=2^{n-1}$, $w_2(n)=n2^{n-1}$	$n2^{n-1}$
5	$w_0(n)=5^n-4^n$, $w_1(n)=w_2(n)=w_3(n)=w_4(n)=4^{n-1}$	$5\cdot4^{n-1}$
6	$w_0(n)=6^n-4^n-3^n+2^n$, $w_1(n)=w_5(n)=2^{n-1}$, $w_2(n)=w_4(n)=4^n/2-2^{n-1}$, $w_3(n)=3^n-2^n$	4^n
7	$w_0(n)=7^n-6^n$, $w_1(n)=w_2(n)=w_3(n)=w_4(n)=w_5(n)$ $=w_6(n)=6^{n-1}$	$9\cdot6^{n-1}$
8	$w_0(n)=8^n-(n^2/2)4^{n-1}-(5n/2)4^{n-1}-4^n$, $w_1(n)=$ $w_3(n)=w_5(n)=w_7(n)=4^{n-1}$, $w_2(n)=w_4(n)=n4^{n-1}$ $w_4(n)=((n^2+n)/2)4^{n-1}$	$(n^2/2+5n/2+9)4^{n-1}$

Table III. Expressions for the Recurrencies and the CAM Storage Requirements of the MODPROD Function.

No. of Values m	No. of Inputs n	MAX(MIN) Function $L_{min}(G_1)=L_{min}(G_2)$	MODSUM Function $L_{min}(G_3)$	MODPROD Function $L_{min}(G_4)$	Uniform Recurrency Distribution
3	2	4	6	4	6
	3	8	18	8	18
	4	16	54	16	54
	5	32	162	32	162
	∞	2^n	$(2/3)3^n$	2^n	$(2/3)3^n$
4	2	10	16	4	16
	3	28	64	12	64
	4	82	256	32	256
	5	244	1,024	80	1,024
	∞	3^n	4^n	$(n/2)2^n$	4^n
5	2	17	25	20	25
	3	65	125	80	125
	4	257	625	320	625
	5	1,025	3,125	1,280	3,125
	∞	4^n	5^n	$(5/4)4^n$	5^n
6	2	29	42	16	42
	3	133	252	64	252
	4	641	1,512	256	1,512
	5	3,157	9,072	1,024	9,072
	∞	5^n	$(7/6)6^n$	4^n	$(7/6)6^n$
7	2	45	63	54	63
	3	243	441	324	441
	4	1,377	3,087	1,944	3,087
	5	8,019	21,609	11,664	21,609
	∞	6^n	$(9/7)7^n$	$(3/2)6^n$	$(9/7)7^n$
8	2	66	96	64	96
	3	408	768	336	768
	4	2,658	6,144	1,728	6,144
	5	17,832	49,152	8,704	49,152
	∞	7^n	$(3/2)8^n$	$(n^2/8)4^n$	$(3/2)8^n$

Table IV. Number of CAM Storage Locations Required in the
CAM Implementation of Specific Multiple-Valued Functions.

V. THE AVERAGE COST OF IMPLEMENTING A MULTIPLE-VALUED FUNCTION.

(a) Direct Calculation of the Average Cost.

The few examples of the previous section show that, when the number of inputs becomes large, there is a wide range of costs over the set of m-valued n-input functions. Thus, it is of interest to compute the average cost over this set, as this is gives a better indication of the costs one can expect in implementing a given function than can be obtained from examples. We proceed as follows. The recurrency distribution of output logic values of some function f corresponds to an m-part partition [5, pp. 41-46] P,

$$P: w_0 + w_1 + \ldots + w_{m-1} = m^n \qquad (8)$$

on m^n. Assume $w_0 \geq w_1 \geq \ldots \geq w_{m-1}$. As there may be consecutive w_i's in (8) which are identical, rewrite P as follows:

$$P: a_1 u_1 + a_2 u_2 + \ldots + a_s u_s = m^n ,$$

where $u_1 > u_2 > \ldots > u_s$ represent the distinct parts of the partition and a_i is the number of occurrences of part u_i. To achieve the minimum cost realization, we represent 0 by 00...0, the next p logic values by 10...0, 01...0, ..., and 00...1, etc.. Note that P provides all the information necessary to compute the total CAM storage requirements. Further, the total CAM storage requirements will be the same for any function in which the recurrency set is the same as specified in (8) except for a rearrangement among input tuples and/or output logic values. The procedure then for calculating the average cost is to enumerate all possible partitions P, compute the storage requirements for each, multiply by the number of functions associated with P, and sum. Dividing this by the total number of functions yields the average cost.

The number of functions N(P) associated with a given partition P is

$$N(P) = \frac{m^n!}{w_0! \, w_1! \, \ldots \, w_{m-1}!} \frac{m!}{(a_1! \, a_2! \, \ldots \, a_s!)} \qquad (9)$$

99

and the average number of CAM storage locations, $AC(m,n)$, needed by an m-valued n-input function is

$$AC(m,n) = (1/m^{m^n}) \sum_{P} N(P) \cdot C(P) , \qquad (10)$$

where $\sum_{P}$ enumerates all m-part partitions on m_n, and $C(P)$ is the cost of partition P. $C(P)$ is the weighted sum,

$$C(P) = \sum_{i=0}^{m-1} w_i x_i , \qquad (11)$$

where $x_0 = 0$, $x_1 = x_2 = \ldots = x_{C(p,1)} = 1$, $x_{C(p,1)+1} = x_{C(p,1)+2} = \ldots = x_{C(p,2)+C(p,1)} = 2$, etc.. (10) was implemented by a computer program in which the function of P was performed by a standard partition enumeration package. The results are shown in Table V.

No. of Values m	No. of Inputs n	Average Number of CAM Locations $AC(m,n)$
2	2	1.250
	3	2.906
3	2	4.241
	3	15.420
4	2	11.566
	3	*
5	2	19.539
	3	*
6	2	33.970
	3	*
7	2	52.652
	3	*

Table V. The Average Number of Storage Locations Required By Multiple-Valued Functions in a CAM Implementation.

The *'s indicate average values which were not computed because of time limitations. Word size also became a limiting factor because of the exponentials.

The form of (10) does not allow insight into how fast the average cost increases and how this compares with the upper bound computed previously. Alternatively, we attack this problem from a different point of view. Instead of the cost associated with m-valued n-input functions, we will consider the costs associated with the recurrency distributions.

(b) The Number of Recurrency Distributions.

Each recurrency distribution is associated with a unique partition of the form given in (8). Thus, we can use established techniques for the enumeration of partitions [5, pp. 41-45]. As an example of the procedure, consider the monotone decreasing recurrency distributions for 3-valued 2-input functions shown below in Table VI. For each of the seven distributions,

w_0	w_1	w_2	Number of CAM Locations Required	Composition of Distribution in Unit Distributions
3	3	3	6	3A
4	3	2	5	2A + B + C
4	4	1	5	A + 3B
5	2	2	4	2A + 3C
5	3	1	4	A + 2B + 2C
6	2	1	3	A + B + 4C
7	1	1	2	A + 6C

Table VI. Monotone Decreasing Recurrency Distributions For 3-valued 2-input Functions.

the cost of the CAM implementation is shown. Each distribution in Table VI is a unique combination of the following "unit" distributions.

Unit Distribution	w_0	w_1	w_2
A	1	1	1
B	1	1	
C	1		

The decomposition of each recurrency distribution into unit distributions is shown in the righthand column of Table VI.

To find all monotone decreasing recurrency distributions, we enumerate the ways to combine the unit distributions and select only those which correspond to 3-valued 2-input functions. Such a process can be performed algebraically. Associate with each unit distribution a generating function on formal variable x, where the exponent represents the contribution to the number of logic variables and the coefficient represents the number of ways so many of the unit distributions can be chosen. That is, for A the corresponding generating function is

$$(x^3 + x^6 + x^9 + \ldots) = x^3(1 - x^3)^{-1} . \qquad (12)$$

The coefficient of x^3, 1, corresponds to the one way a single unit distribution A contributes to w_0, w_1, and w_2, while the exponent 3 corresponds to the amount of the contribution. x^6 corresponds to the one way two unit distributions A contribute 6 to w_0, w_1, and w_2, etc.. Similarly, the generating

functions corresponding to the unit distributions B and C are

$$(1 + x^2 + x^4 + \ldots) = (1 - x^2)^{-1} \qquad (13)$$

$$\text{and } (1 + x + x^2 + \ldots) = (1 - x)^{-1}, \qquad (14)$$

respectively. A constant 1 appears in (13) and (14) because of the possibility that the corresponding unit distribution does not appear in the composite distribution. Taking the product of the generating functions (12), (13), and (14) yields

$$(x^3 + x^6 + x^9 \ldots)(1 + x^2 + x^4 + \ldots)(1 + x + x^2 + \ldots) =$$
$$x^3(1 - x^3)^{-1}(1 - x^2)^{-1}(1 - x)^{-1}, \quad (15)$$

the generating function for monotone decreasing recurrency distributions of 3-valued 2-input functions. (15), it turns out, is the generating function for the number of partitions on 1, 2, and 3 having at least one 3. The coefficient of x^i in (15) is the number of ways the unit distributions can be combined to form a composite recurrency distribution with i output levels. For example, expanding (15) yields

$$x^3(1 - x^3)^{-1}(1 - x^2)^{-1}(1 - x)^{-1} = x^3 + x^4 + 2x^5 + 3x^6$$
$$+ 4x^7 + 5x^8 + 7x^9 + \ldots \quad (16)$$

The coefficient of x^9, 7, is the number of monotone decreasing recurrency distributions of functions with 9 ($=3^2$) output logic values, i.e. the 7 3-valued 2-input functions listed in Table VI. Similarly, the number of monotone decreasing recurrency distributions of 3-valued 3-input functions is the coefficient of x^{27} ($27 = 3^3$) in (16).

An expression for the value of the coefficient of x^i in (16) can be obtained by standard combinatoric methods [6, pp. 83-99]. Alternatively, a simpler closed-form expression can be derived which is accurate for large i by the methods shown in Bender[7]. We prefer the second approach.

Let b_i be the coefficient of x^i in (16). Applying Theorem 4 of [7] to (16) yields

$$b_i \sim i^2/12 .$$

Since we are only interested in those values of i corresponding to the number of input tuples associated with a 3-valued n-input functions, i = 3^n, and the number of monotone decreasing recurrency distributions is asymptotically equal to

$$9^n/12.$$

In a similar manner, the asymptotic approximations for other values of m can be obtained. Table VII shows the generating functions and asymptotic approximations for the number of monotone decreasing recurrency distributions for $3 \leq m \leq 7$.

Number of Logic Values m	Generating Functions	Asymptotic Approx. For the No. of Monotone Recurrency Distr.
3	$x^3(1-x^3)^{-1}(1-x^2)^{-1}(1-x)^{-1} = G_3$	$(3^2)^n/12$
4	$(x^4/x^3)(1-x^4)^{-1}G_3 = G_4$	$(4^3)^n/144$
5	$(x^5/x^4)(1-x^5)^{-1}G_4 = G_5$	$(5^4)^n/2,880$
6	$(x^6/x^5)(1-x^6)^{-1}G_5 = G_6$	$(6^6)^n/86,400$
7	$(x^7/x^6)(1-x^7)^{-1}G_6 = G_7$	$(7^6)^n/3,628,000$

Table VII. Generating Functions and Asymptotic Approximations For the Number of Monotone Decreasing Recurrency Distributions Among m-Valued n-Input Functions.

(c) The Average Cost Over the Monotone Decreasing Recurrency Distributions.

The average number of CAM storage locations over the monotone decreasing recurrency distributions can be calculated using a modified version of the generating functions described above by including another formal variable y. That is, every occurrence of x in the enumeration will be accompanied by y^j, where j is the number of CAM locations required. Note that units A, B, and C contribute 2, 1, and 0, respectively, to the overall cost of the distribution of which they are a part. Therefore, instead of (12), use

$$(x^3y^2 + x^6y^4 + x^9y^6 + \ldots) = x^3y^2(1 - x^3y^2)^{-1}, (17)$$

and instead of (13), use

$$(1 + x^2y + x^4y^2 + \ldots) = x^2y(1 - x^2y)^{-1} . (18)$$

Because unit C contributes nothing to the overall cost, (14) will be used without modification. The generating function which accounts for the cost is then the product of (17), (18), and (14) or

$$x^3y^2(1-x^3y^2)^{-1}(1 - x^2y)^{-1}(1 - x)^{-1}. \qquad (19)$$

Expanding (19) and collecting terms containing x^i shows the distribution of costs over the recurrency distributions. For example, with respect to x^9, we have from (19) the term

$$(y^2 + y^3 + 2y^4 + 2y^5 + y^6)x^9. \qquad (20)$$

This shows that there is one monotone decreasing recurrency distribution with cost 2 (y^2), one with cost 3 (y^3), two with cost 4 ($2y^4$), two with cost 5 ($2y^5$), and one with cost 6 (y^6). The result from (20) agrees with Table VI, as it should.

If we differentiate (19) with respect to y and set y to 1, terms of the form (20) become weighted

sums which, when divided by the number of monotone decreasing recurrency distributions, yield the average cost per distribution. In computing the ith coefficient, again we prefer to use the asymptotic methods of Bender[7]. Table VIII shows the average cost as calculated by this method as well as the highest cost associated with the uniform recurrency distribution as calculated previously.

vol. 66, pp. 1240-1255, October 1978.

[3] Z. G. Vranesic and S. G. Zaky, "Multivalued logic in local digital intercommunication systems," Proc. COMPCON 1982 Fall 1982, pp. 127-134, September 1982.

[4] C. Papachristou, "Content-addressable memory requirements for multivalued logic," Proc. of

Number of Output Logic Values m	Average Number of CAM Storage Locations Over the Monotone Decreasing Recurrency Distributions For Large n	Highest Cost (Uniform Recurrency Distributions)	Ratio of Average Cost to Highest Cost For Large n
3	$.5556 \cdot 3^n$	$2 \cdot 3^{n-1}$	.8333
4	$.5417 \cdot 4^n$	$4 \cdot 4^{n-1}$	.5417
5	$.5832 \cdot 5^n$	$5 \cdot 5^{n-1}$	.5832
6	$.6806 \cdot 6^n$	$7 \cdot 6^{n-1}$	.5833
7	$.7670 \cdot 7^n$	$9 \cdot 7^{n-1}$	.5966

Table VIII. Average Number of CAM Storage Locations Over the Monotone Decreasing Recurrency Distributions For Large n.

VI. CONCLUDING REMARKS.

The righthand column of Table VIII shows the ratio of the average cost of CAM implemented m-valued n-input functions over the recurrency distributions to the highest cost, when n is large. For $m \geq 4$, the ratio is between 0.5 and 0.6. Although this may not be representative if one uses the average over all functions rather than over the recurrency distributions, the results of Section V indicate that the two averages are comparable, at least for small values of m and n. If indeed the ratios of Table VIII are representative, the upper bound storage requirements are reasonably close to the storage requirements of a random function.

However, there is the question of whether a circuit designer tends to synthesize random functions or whether the functions chosen tend to be either more or less costly than the average. Specific functions considered in Section IV occupy both ends of the spectrum. The MODSUM is among the most expensive m-valued n-input functions, while the MODPROD is among the least expensive.

REFERENCES

[1] M. Stark, "Two bits per cell ROM," Digest of Papers: COMPCON 1981 Spring, pp. 209-212, February 1981.

[2] J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak, and C. B. Weinstock, "SIFT: Design and analysis of a fault tolerant computer for aircraft control," Proc. IEEE,

the Eleventh Inter. Symp. on Multiple-Valued Logic, pp. 62-72, May 1981.

[5] C. L. Liu, Introduction to Combinatorial Mathematics, McGraw-Hill, New York, 1968.

[6] A. Tucker, Applied Combinatorics, Wiley, New York, 1980.

[7] E. A. Bender, "Asymptotic methods in enumeration," SIAM Review, pp. 485-515, October 1974.

A FAST COMPLEMENTATION ALGORITHM FOR SUM-OF-PRODUCTS EXPRESSIONS OF MULTIPLE-VALUED INPUT BINARY FUNCTIONS

Tsutomu Sasao

Mathematical Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

and

Department of Electronics Engineering
Osaka University
Osaka 565, Japan

ABSTRACT: A recursive algorithm to obtain a complement of a sum-of-products expression for a binary function of p-valued input variables is presented. It produces at most $p^n/2$ products for n-variables functions, whereas an elementary algorithm produces $O(t^n \cdot n^{(t-1)/2})$ products where $t = 2^p - 1$. It is 10 ~ 30 times faster then the elementary one when p=2 and n=8.

I. Introduction:

As an elementary method to obtain a complement of a sum-of-products expression for f, the following is well known.

1. By using De Morgan's law, obtain a product-of-sums expression for $\bar{f}$.

2. By using the distributive law, obtain a sum-of-products expression for $\bar{f}$.

3. By using the absorption law, simplify the sum-of-products expression.

However, this method becomes quite inefficient when the number of input variables is large, because it will produce all the prime implicants of $\bar{f}$. For example, the elementary method will generate $O(3^n/n)$ products for a class of n-variable switching functions (two-valued input binary functions) [1], whereas the presented algorithm will generate at most 2^{n-1} products. The new algorithm is about 10 ~ 30 times faster than the elementary one for switching functions of 8-variables.

Binary functions are useful in designing programmable logic arrays with decoders [2] and other circuits [3], [4]. Simplification of the expressions for the binary functions will reduce the complexities of circuits. A fast complementation algorithm has been desired because practical minimization algorithms such as MINI [5] and ESPRESSO [6] require the complement of the given function.

The proposed algorithm has been incorporated into MINI and other systems and has been effectively used to design logical circuits.

II. Definitions and an Elementary Method for Complementation

Definition 2.1[4]: A mapping $\overset{n}{\underset{i=1}{\times}} P_i \to B$ is called a multiple-valued input **binary function**, where $P_i = \{0,1,..., p_i-1\}$, and $B = \{0, 1\}$.

Definition 2.2: Let X_i be a variable on P_i. $X_i^{S_i}$ is a **literal** of X_i when $S_i \subseteq P_i$.
$X_i^{S_i}$ represent a function

$$X_i^{S_i} = \begin{cases} 0 & \text{if } X_i \notin S_i \\ 1 & \text{if } X_i \in S_i \end{cases}$$

Definition 2.3: A product of literals $X_1^{S_1} \cdot X_2^{S_2} \cdot ... \cdot X_n^{S_n}$ is called a **product**. A sum of products is called a **sum-of-products expression**.

Theorem 2.1[2]: An arbitrary binary function $\overset{n}{\underset{i=1}{\times}} P_i \to B$ can be represented by a sum-of-products expression

$$f(X_1, X_2,..., X_n) = \underset{(S_1,S_2,...,S_n)}{\vee} X_1^{S_1} \cdot X_2^{S_2} \cdot ... \cdot X_n^{S_n}$$

where $S_i \subseteq P_i$.

Definition 2.4: Let E be a product. E is called a **prime implicant** if $E \leq f$ and E is maximal (i.e. there is no E_1 such that $E, E_1 \leq f$).

Lemma 2.1: Let f, g and h be binary functions.

$\overline{f \cdot g} = \bar{f} \vee \bar{g}$ (De Morgan's law)

$(f \vee g) \cdot h = f \cdot h \vee g \cdot h$ (Distributive law)

$f \vee f \cdot g = f$ (Absorption law)

As an elementary method to obtain a complement of

sum-of-products expression, the following is well known.

Algorithm 2.1:

1. By using De Morgan's law, convert a complement of a given expression into a product-of-sums form.

2. By using the distributive law, expand the expression into a sum-of-products form. Delete null products (If $A \cap B = \phi$ then $X^A \cdot X^B \equiv \phi$) and redundant literals (If $A \supseteq B$ then $X^A \cdot X^B = X^B$).

3. By using the absorption law, drop subsuming products $(p \lor pq = p)$.

Example 2.1:

Consider a binary function

$$f: \{0,1\} \times \{0,1,2\} \times \{0,1,2,3\} \to \{0,1\}$$

and an expression

$$\mathcal{F} = X_1^0 \cdot X_2^1 \cdot X_3^{\{1,3\}} \lor X_1^1 \cdot X_2^{\{0,2\}} \cdot X_3^{\{1,2\}} \lor X_2^{\{1,2\}} \cdot X_3^1.$$

Let's obtain a complement of $\mathcal{F}$ by Algorithm 2.1. First, by De Morgan's law, convert it into a product-of-sums form.

$$\overline{\mathcal{F}} = (X_1^1 \lor X_2^{\{0,2\}} \lor X_3^{\{0,2\}}) \cdot (X_1^0 \lor X_2^1 \lor X_3^{\{0,3\}}) \cdot (X_2^0 \lor X_3^{\{0,2,3\}})$$

Second, by the distributive law, we have the following:

$$\overline{\mathcal{F}} = (X_1^1 \cdot X_2^1 \lor X_1^1 \cdot X_3^{\{0,3\}} \lor X_1^0 \cdot X_2^{\{0,2\}} \lor X_2^{\{0,2\}} \cdot X_3^{\{0,3\}} \lor X_1^0 \cdot X_3^{\{0,2\}} \lor X_2^1 \cdot X_3^{\{0,2\}} \lor X_3^0) \cdot (X_2^0 \lor X_3^{\{0,2,3\}})$$

In the above expression, $X_1^1 \cdot X_1^0$ and $X_2^{\{0,2\}} \cdot X_2^1$ etc. are omitted because they are null products. By using the distributive law again, we have the sum-of-products expression:

$$\overline{\mathcal{F}} = X_1^1 \cdot X_2^0 \cdot X_3^{\{0,3\}} \lor X_1^0 \cdot X_2^0 \lor X_2^0 \cdot X_3^{\{0,3\}} \lor X_1^0 \cdot X_2^0 \cdot X_3^{\{0,2\}}$$
$$\lor X_2^0 \cdot X_3^0 \lor X_1^1 \cdot X_2^1 \cdot X_3^{\{0,2,3\}}$$
$$\lor X_1^1 \cdot X_3^{\{0,3\}} \lor X_1^0 \cdot X_2^{\{0,2\}} \cdot X_3^{\{0,2,3\}}$$
$$\lor X_2^{\{0,2\}} \cdot X_3^{\{0,3\}} \lor X_1^0 \cdot X_3^{\{0,2\}} \lor X_2^1 \cdot X_3^{\{0,2\}} \lor X_3^{\{0\}}$$

Third, by the absorption law, we can delete products $X_1^1 \cdot X_2^0 \cdot X_3^{\{0,3\}}$ and $X_2^0 \cdot X_3^0$ etc., because $X_2^0 \cdot X_3^{\{0\}} \leq X_3^0$ and $X_1^1 \cdot X_2^0 \cdot X_3^{\{0,3\}} \leq X_1^1 \cdot X_3^{\{0,3\}}$. Hence, we have the sum-of-

products expression:

$$\overline{\mathcal{F}} = X_1^0 \cdot X_2^0 \lor X_1^1 \cdot X_2^1 \cdot X_3^{\{0,2,3\}} \lor X_1^1 \cdot X_3^{\{0,3\}}$$
$$\lor X_1^0 \cdot X_2^{\{0,2\}} \cdot X_3^{\{0,2,3\}} \lor X_2^{\{0,2\}} \cdot X_3^{\{0,3\}} \lor X_1^0 \cdot X_3^{\{0,2\}}$$
$$\lor X_2^1 \cdot X_3^{\{0,2\}} \lor X_3^{\{0\}}$$

Note that $\overline{\mathcal{F}}_1$ contains 8 products.

(End of Example)

Straightforward application of Algorithm 2.1 is quite inefficient. It might be made more efficient by the simplifying the intermediate results by using absorption law or by changing the order of expansion. However, in any case, Algorithm 2.1 will generate many products. This is because that Algorithm 2.1 will generate all the prime implicants of $\bar{f}$, which is stated by the following theorem.

Theorem 2.1: Let $\mathcal{F}$ be a sum-of-products expression of a binary function f, and $\overline{\mathcal{F}}$ be an expression obtained by using Algorithm 2.1. Then, $\overline{\mathcal{F}}$ contains all the prime implicants of $\bar{f}$.

(Proof). Similar to the switching function [7]

(Q.E.D)

III. A Fast Complementation Algorithm

By extending Shannon's expansion theorem to binary functions and applying the complementation theorem of Hong-Ostapko [9], we have the following:

Lemma 3.1: Let a binary function be represented by

$$f = X^0 \cdot f_0 \lor X^1 \cdot f_1 \lor \ldots \lor X^{p-1} \cdot f_{p-1}.$$

Then a complement of f is given by

$$\bar{f} = X^0 \bar{f}_0 \lor X^1 \cdot \bar{f}_1 \lor \ldots \lor X^{p-1} \cdot \bar{f}_{p-1}.$$

where $f_i = f(X \leftarrow i)$.

By using Lemma 3.1 recursively, we can obtain a complement of an expression. It is possible to make an algorithm to generate at most $\prod_{i=1}^{n} p_i / (\max\{p_i\})$ products. However, experiments showed that a program simply based on Lemma 3.1 was not so fast for large practical problems. One reason for it is that most variables appear in a small number of products (i.e., a lot of "don't cares" in the array). Therefore, for the practical problems, the following algorithm has been developed.

Algorithm 3.1: Let $\mathcal{F}$ be a given expression. Use the following rules recursively.

Rule 1. If $\mathcal{F}$ is a constant:

If $\mathcal{F} = 1$, then $\overline{\mathcal{F}} = 0$

If $\mathscr{F} = 0$, then $\overline{\mathscr{F}} = 1$.

Rule 2. If $\mathscr{F}$ depends on only one-variable: i.e. if

$\mathscr{F} = X_1^{s_a} \vee X_1^{s_b} \vee ... \vee X_1^{s_z}$ then $\overline{\mathscr{F}} = X_1^{\bar{s}}$ where

$S = S_a \cup S_b \cup ... \cup S_z$.

Rule 3. If $\mathscr{F}$ consists of one product i.e., if

$\mathscr{F} = X_1^{S_1} \cdot X_2^{S_2} ... X_l^{S_l}$ then

$\overline{\mathscr{F}} = X_1^{\bar{s}} \vee X_1^{s_1} \cdot X_2^{\bar{s}_2} \vee ... \vee X_1^{s_1} \cdot X_2^{s_2} ... X_{l-1}^{s_{l-1}} \cdot X_l^{\bar{s}_l}$

Rule 4. If $\mathscr{F}$ has a common factor, i.e. if $\mathscr{F}$ can be written as

$\mathscr{F} = X_1^{S_1} X_2^{S_2} ... X_l^{S_l} \cdot \mathscr{G}$.

by renaming variables, where $\mathscr{G}$ does not contain variables $X_1, X_2, ..., X_l$, then

$\overline{\mathscr{F}} = X_1^{\bar{s}_1} \vee X_1^{s_1} X_2^{\bar{s}_2} \vee ... \vee X_1^{s_1} X_2^{s_2} ... X_l^{\bar{s}_l} \; \overline{\mathscr{G}}$.

Rule 5. If $\mathscr{F}$ can be decomposed with a variable X_i, i.e. if $\mathscr{F}$ can be written as

$\mathscr{F} = X_i^0 \cdot \mathscr{G}_0 \vee X_i^1 \cdot \mathscr{G}_1 \vee ... \vee X_i^{p_i-1} \cdot \mathscr{G}_{p_i-1}$.

then

$\overline{\mathscr{F}} = X_i^0 \overline{\mathscr{G}}_0 \vee X_i^1 \overline{\mathscr{G}}_1 \vee ... \vee X_i^{p_i-1} \overline{\mathscr{G}}_{p_i-1}$.

where $\mathscr{G}_k$ ($k = 0, ..., p_i-1$) do not contain the variable X_i.

Rule 6. Otherwise, $\mathscr{F}$ can be written as

$\mathscr{F} = X_1^{S_1} \cdot X_2^{S_2} \cdot ... \cdot X_l^{S_l} \vee \mathscr{G}$

by renaming the variables. Then $\overline{\mathscr{F}}$ is given by

$\overline{\mathscr{F}} = X_1^{\bar{s}_1} \overline{\mathscr{G}}_1 \vee X_1^{s_1} X_2^{\bar{s}_2} \overline{\mathscr{G}}_2 \vee ... \vee X_1^{s_1} X_2^{s_2} ... X_{l-1}^{s_{l-1}} X_l^{s} \overline{\mathscr{G}}_l$,

where $\mathscr{G}_i$ is obtained from $(X_1^{s_1} \cdot X_2^{s_2} \cdot ... \cdot X_i^{\bar{s}_i}) \wedge \mathscr{G}$ by deleting null products.

Definition 3.1: A sum-of-products expression is disjoint if all products are mutually disjoint, i.e.,

$\mathscr{F} = \alpha_1 \vee \alpha_2 \vee ... \vee \alpha_s$,
$\alpha_i \cdot \alpha_j = 0 \; (i \neq j)$ or $s = 1$

Theorem 3.1: Algorithm 3.1 generates disjoint sum-of-products expression for f. The number of products in $\mathscr{F}$ is

denoted by $t(\mathscr{F})$. By Theorem 3.1, it is clear that

$$t(\overline{\mathscr{F}}) \leq \prod_{i=1}^{n} p_i$$

where $\overline{\mathscr{F}}$ is obtained by Algorithm 3.1.

Theorem 3.2: Let $\mathscr{F}_n$ be a sum-of-products expression for

$$f_n: \mathop{\times}_{i=1}^{n} P_i \to B.$$

Let $\overline{\mathscr{F}}_n$ be an expression obtained by Algorithm 3.1, then

$$t(\overline{\mathscr{F}}_n) \leq \frac{1}{2} \prod_{i=1}^{n} p_i$$

(Proof). Proof will be done by the induction on n and restriction of f_n.

Rule 1. When $n = 0$: $t(\overline{\mathscr{F}}_0) \leq 1$ and the theorem holds for $n = 0$.

Rule 2. When $n = 1$: $t(\overline{\mathscr{F}}_1) \leq 1$ and the theorem hold for $n = 1$.

Rule 3. When $\mathscr{F}$ consists of one product: $X_1^{S_1} \cdot X_2^{S_2} \cdot ... \cdot X_l^{S_l}$. Then, $t(\overline{\mathscr{F}}_n) = l \leq n \leq 2^{n-1}$, and the theorem holds $(n \geq 2)$.
From here, suppose that the theorem holds for $n-1, n-2, ..., 1, 0$, and for the restriction of f_n.

Rule 4. When $\mathscr{F}_n$ has a common factor, i.e. $\mathscr{F}$ can be written as follows by renaming the variables:

$\mathscr{F}_n = X_1^{S_1} \cdot X_2^{S_2} \cdot ... \cdot X_l^{S_l} \cdot \mathscr{G}$
$t(\overline{\mathscr{F}}_n) = t(X_1^{\bar{s}_1} \vee X_1^{s_1} \cdot X_2^{\bar{s}_2} \vee ... \vee X_1^{s_1} \cdot X_2^{s_2} \cdot ... \cdot X_l^{\bar{s}_l}) + t(\overline{\mathscr{G}})$.

Since $\mathscr{G}$ does not contain the variables $X_1, X_2, ..., X_l$, it has at most $(n-l)$ variables. By the hypothesis of induction $t(\overline{\mathscr{G}}) \leq \frac{1}{2} \prod_{i=l+1}^{n} p_i$. Hence $t(\overline{\mathscr{F}}) \leq l + \frac{1}{2} \prod_{i=l-1}^{n} p_i \leq \frac{1}{2} \prod_{i=1}^{n} p_i$ and the theorem holds.

Rule 5. When $\mathscr{F}_n$ can be decomposed with respect to X_i, i.e. $\mathscr{F}_n$ can be written as

$\mathscr{F}_n = X_1^0 \cdot \mathscr{G}_0 \vee X_1^1 \cdot \mathscr{G}_1 \vee ... \vee X_1^{p_1-1} \cdot \mathscr{G}_{p_1-1}$

by renaming the variables:

$$t(\mathscr{F}_n) = \sum_{i=0}^{p_1-1} t(\mathscr{G}_i).$$

By the hypothesis of induction $t(\mathscr{G}_i) \leq \frac{1}{2} \prod_{j=2}^{n} p_j$ Hence $t(\mathscr{F}_n) \leq p_1 \times \frac{1}{2} \prod_{j=2}^{n} p_j = \frac{1}{2} \prod_{j=1}^{n} p_j$ and the theorem holds.

Rule 6. Otherwise, $\mathscr{F}$ can be written as

$$\mathscr{F} = X_1^{S_1} \cdot X_2^{S_2} \cdot \ldots \cdot X_l^{S_l} \vee \mathscr{G}$$

by renaming the variables. Because

$$\bar{\mathscr{F}} = X_1^{\bar{S}_1} \cdot \bar{\mathscr{G}}_1 \vee X_1^{S_1} \cdot X_2^{\bar{S}_2} \cdot \bar{\mathscr{G}}_2 \vee \ldots \vee X_1^{S_1} \cdot X_2^{S_2} \cdot \ldots \cdot X_l^{\bar{S}_l} \cdot \bar{\mathscr{G}}_l,$$

$$t(\bar{\mathscr{F}}) = \sum_{i=1}^{l} t(\bar{\mathscr{G}}_i).$$

$\mathscr{G}_i$ is obtained from $(X_1^{S_1} \cdot X_2^{S_2} \cdot \ldots \cdot X_l^{\bar{S}_l}) \wedge \mathscr{G}$ by deleting null products, and has a common factor $X_1^{S_1} \cdot X_2^{S_2} \cdot \ldots \cdot X_k^{\bar{S}_k}$. Let $|S_k| = a_k$, where $1 \le a_k \le p_k - 1$. In $\mathscr{G}_i$, X_k takes at most a_k distinct values; in other words, $\mathscr{G}_i$ represents a restriction of f_n:

$$\{0,1,\ldots,a_1-1\} \times \{0,1,\ldots,a_2-1\} \times \ldots \times \{0,1,\ldots,(p_i-a_i)-1\}$$
$$\times \mathop{\times}_{k=i+1}^{n} P_k \to B$$

By the hypothesis of induction,

$$t(\mathscr{G}_i) \le \frac{1}{2} \left(\prod_{k=1}^{i-1} a_k \right) \cdot (p_i - a_i) \times \left(\prod_{k=i+1}^{n} P_k \right).$$

Let $b_i = a_i / p_i$ and we have

$$t(\bar{\mathscr{G}}_i) \le \frac{1}{2} \prod_{k=1}^{n} P_k \cdot \left(\prod_{k=1}^{i-1} b_k \right) \times (1 - b_i)$$

Hence

$$t(\bar{\mathscr{F}}) \le$$
$$\frac{1}{2} \prod_{k=1}^{n} p_k \{ (1-b_1) + b_1(1-b_2) + \ldots + b_1 b_2 \ldots b_{l-1}(1-b_l) \}$$
$$\le \frac{1}{2} \prod_{k=1}^{n} p_k$$

and the theorem holds.

We have exhausted all possible cases and proved the theorem by induction.

(Q.E.D)

In Rule 6 of Algorithm 3.1, the selection of the products and the ordering of the variables influence the efficiency of the algorithm. After doing a lot of experiments on practical circuits, we use the following heuristics.

Heuristic 3.1:

1. Which product to select: Choose one with the least number of literals (i.e., the number of literals such that $|S_i| \ne p_i$). If a tie occurs, choose one with maximal $\sum_{i=1}^{l} |S_i|$, where $X_1^{S_1} \cdot X_2^{S_2} \cdot \ldots \cdot X_l^{S_l}$.

2. The ordering of variables: Expand a product in the ascending order of $|S_i|/p_i$. If a tie occurs, expand first using the variable with the smallest $\alpha_i = $ (sum of number of 1's of each part in bit representation of $\mathscr{G}$)/p_i.

Example 3.1: Consider the function shown in Example 2.1:

$$\mathscr{F} = X_1^0 \cdot X_2^1 \cdot X_3^{\{1,3\}} \vee X_1^1 \cdot X_2^{\{0,2\}} \cdot X_3^{\{1,2\}} \vee X_2^{\{1,2\}} \cdot X_3^1$$

First use Rule 6. $X_2^{\{1,2\}} \cdot X_3^1$ is a product with least number of literals. $\mathscr{F}$ is written as follows:

$$\mathscr{F} = X_2^{\{1,2\}} \cdot X_3^1 \vee \mathscr{G}, \text{ where}$$
$$\mathscr{G} = X_1^0 \cdot X_2^1 \cdot X_3^{\{1,3\}} \vee X_1^1 \cdot X_2^{\{0,2\}} \cdot X_3^{\{1,2\}}$$

Because $\dfrac{|S_2|}{p_2} = \dfrac{2}{3}$ and $\dfrac{|S_3|}{p_3} = \dfrac{1}{4}$, expand it in the order of X_3 and X_2.

$$\bar{\mathscr{F}} = (X_3^{\{0,2,3\}} \vee X_2^0 \cdot X_3^1) \cdot \bar{\mathscr{G}} = X_3^{\{0,2,3\}} \cdot \bar{\mathscr{G}}_1 \vee X_2^0 \cdot X_3^1 \cdot \bar{\mathscr{G}}_2$$

where $\mathscr{G}_1 = X_3^{\{0,2,3\}} \cdot \mathscr{G} = X_1^0 \cdot X_2^1 \cdot X_3^3 \vee X_1^1 \cdot X_2^{\{0,2\}} \cdot X_3^2$ and
$\mathscr{G}_2 = X_2^0 \cdot X_3^1 \cdot \mathscr{G} = X_1^0 \cdot X_2^0 \cdot X_3^1$.

Next let's obtain $\bar{\mathscr{G}}_1$ and $\bar{\mathscr{G}}_2$ recursively. $\mathscr{G}_1$ can be written as $\mathscr{G}_1 = X_1^0 \cdot (X_2^1 \cdot X_3^3) \vee X_1^1 \cdot (X_2^{\{0,2\}} \cdot X_3^2)$. $\mathscr{G}_1$ can be decomposed with respect to X_1. By Rule 5, we have

$$\bar{\mathscr{G}}_1 = X_1^0 \cdot (X_2^{\{0,2\}} \vee X_2^1 \cdot X_3^{\{0,1,2\}}) \vee X_1^1 \cdot (X_2^1 \vee X_2^{\{0,2\}} \cdot X_3^{\{0,1,3\}}).$$

$\mathscr{G}_2$ consists of one product and by Rule 3,
$$\bar{\mathscr{G}}_2 = X_1^1 \vee X_1^0 \cdot X_2^{\{1,2\}} \vee X_1^0 \cdot X_2^0 \cdot X_3^{\{0,2,3\}}.$$

Hence

$$\bar{\mathscr{F}} = X_3^{\{0,2,3\}} \cdot \{ X_1^0 \cdot (X_2^{\{0,2\}} \vee X_2^1 \cdot X_3^{\{0,1,2\}})$$
$$\vee X_1^1 \cdot (X_2^1 \vee X_2^{\{0,2\}} \cdot X_3^{\{0,1,3\}}) \}$$
$$\vee X_2^0 \cdot X_3^1 \cdot \{ X_1^1 \vee X_1^0 \cdot X_2^{\{1,2\}} \vee X_1^0 \cdot X_2^0 \cdot X_3^{\{0,2,3\}} \}$$
$$X_1^0 \cdot X_2^{\{0,2\}} \cdot X_3^{\{0,2,3\}} \vee X_1^0 \cdot X_2^0 \cdot X_3^{\{0,2\}}$$
$$= \vee X_1^1 \cdot X_2^1 \cdot X_3^{\{0,2,3\}} \vee X_1^1 \cdot X_2^{\{0,2\}} \cdot X_3^{\{0,3\}}$$
$$\vee X_1^1 \cdot X_2^0 \cdot X_3^1$$

Note that $\bar{\mathscr{F}}$ contains 5 products.

(End of Example)

IV. Experimental Results:

Algorithm 3.1 has been programmed in APL and compared with other algorithms written in APL.

1. Table 4.1 shows the comparison of Algorithm 2.1 (U_n # F), the disjoint sharp algorithm of MINI [5], and Algorithm 3.1. U_n denotes a universal cube. (U_n # $\mathscr{F}$) can be considered as an implementation of Algorithm 2.1. Ⓝ is similar to #, but will generate disjoint sum-of-products expressions.

106

First, truth tables for 8-variable switching functions were randomly generated. Then, the functions were simplified by the distance-one-merge algorithm [5]. $t(\mathscr{F})$ denotes the number of products in a simplified expression. Lastly, the complement of the expressions were obtained. $t(\overline{\mathscr{F}})$ denotes the number of products in the complement $\overline{\mathscr{F}}$. Table 4.1 shows that the disjoint sharp (#) algorithm and Algorithm 3.1 are 10 ~ 30 times faster than algorithm 2.1 (D # F) and will generate simpler expressions. (See the entries for $n = 8$ and $p = 2$.) Also, the truth tables of 8-variable switching functions were decoded to make 4-variable binary functions of 4-valued variables. Also in this case, Disjoint sharp and Algorithm 3.1 were faster and produced simpler expressions. (See the entries for $n = 4$ and $p = 4$.)

2. Table 4.2 shows the comparison of Disjoint sharp (U_n(#)F) and Algorithm 3.1. Control circuits for microprocessors were used to compare the performance of two algorithms. For example see the entries for D2. D2 is an 8-input 7-output circuit. A characteristic function for a two-level

PLA [2] is a mapping

f: $P^8 \times M \to B$; P = {0,1}, M = {0,1,...,6}, B = {0,1}

A simplified expression $\mathscr{F}$ for f has 43 products. Also, a characteristic function for a PLA with two-bit decoders [2] is a mapping.

f: $P^4 \times M \to B$; P = {0,1,2,3}, M = {0,1,...,6}, B = {0,1}

A simplified expression $\mathscr{F}$ for f has 42 products. Table 4.2 shows that Algorithm 3.1 generates simpler solutions (fewer products) than U_n(#)F. This is a desirable property because in MINI, (U_n(#)F) often produces an excessive number of products which prevents completing the initial phase of computing the complement for large problems.

3. Recently, R.K. Brayton et. al have independently developed a fast complementation algorithm [13]. It is for ordinary multiple-output switching functions only, and cannot treat multiple-valued variables. It is difficult to compare the performance of their algorithm with Algorithm 3.1 because of different data structures. In most cases, Algorithm 3.1 produced comparable solutions, but took longer time.

Table 4.1: Numbers of products in complement expressions and their computation time for Sharp,

| | | | Algorithm 2.1 | | Disjoint Sharp | | Algorithm 3.1 | |
| | | | U_n # F | | U_n(#)F | | | |
	U	$t(\mathscr{F})$	CPU time (sec)	$t(\overline{\mathscr{F}})$	CPU time (sec)	$t(\overline{\mathscr{F}})$	CPU time (sec)	$t(\overline{\mathscr{F}})$
	32	23	38.814	171	2.098	67	1.167	51
p = 2	64	39	57.631	203	3.186	82	4.493	68
n = 8	96	57	65.232	163	4.191	87	2.925	79
	128	57	59.472	116	4.239	73	2.930	63
	32	21	15.020	243	0.701	47	0.735	41
p = 4	64	33	22.494	207	1.436	54	0.883	56
n = 4	96	45	43.494	204	2.485	56	1.647	54
	128	53	29.927	131	2.583	56	1.624	56

$f: P^n \to B$; P = {0,1,..., p – 1} u = $|f^{-1}(1)|$.

$\mathscr{F}$: sum-of-products expression for f; $\overline{\mathscr{F}}$: sum-of-products expression for $\bar{f}$.

$t(\mathscr{F})$: Number of products in $\mathscr{F}$; $t(\overline{\mathscr{F}})$: number of products in $\overline{\mathscr{F}}$.

Table 4.2: Numbers of products in complement expressions and their computation time for

Disjoint sharp and Algorithm 3.1.

					Disjoint Sharp $U_n \textcircled{\#} F$		Algorithm 3.1	
Circuit name	n	p	m	$t(\mathscr{F})$	CPU Time (sec)	$t(\bar{\mathscr{F}})$	CPU Time (sec)	$t(\bar{\mathscr{F}})$
D2	8	2	7	43	1.904	125	1.548	67
	4	4	7	42	1.166	106	1.569	95
R1	8	2	31	33	1.652	123	1.231	31
	4	4	31	32	1.016	63	.976	40
I1	16	2	17	110	10.162	333	5.429	202
	8	4	17	103	5.779	288	4.720	229
I4	32	2	20	222	64.954	3042	23.375	651
	16	4	20	204	33.841	1633	32.038	1148
I5	24	2	14	62	8.783	918	7.460	255
	12	4	14	61	5.287	1100	19.353	667
A2	10	2	8	89	5.372	228	6.417	188
	5	4	8	83	3.124	216	5.417	165

$f: P^n \times M \to B$, $P = \{0,1,..., p-1\}$ $M = \{0,1,..., m-1\}$

$\mathscr{F}$: sum-of-products expression for f; $\bar{\mathscr{F}}$: sum-of-products expression for $\bar{f}$.

$t(\mathscr{F})$: Number of products in $\mathscr{F}$; $t(\bar{\mathscr{F}})$: number of products in $\bar{\mathscr{F}}$.

108

V. Conclusions

1. The elementary method to obtain the complement of sum-of-product expression for f will generate all the prime implicants of $\bar{f}$, and is quite inefficient.

2. The average number of prime implicants for binary functions $\{0,1,...,p-1\}^n \rightarrow B$ is larger than $1/2 \, p^n$ for large n.

3. Algorithm 3.1 will generate at most $1/2 \, p^n$ products. It is $10 \sim 30$ times faster than the elementary one when $n = 8$ and $p = 2$.

4. Algorithm 4.1 produces fewer products than the disjoint sharp algorithm used by MINI for large practical problems.

Acknowledgement

References

[1] B. Dunham and R. Fridshal, "The problem of simplifying logical expressions", Journal of Symbolic Logic, Vol. 24, pp. 17-19, 1959.

[2] T. Sasao, "Multiple-valued decomposition of generalized Boolean functions and the complexity of programmable logic arrays", IEEE Trans. on Comput., Vol. C-30, No. 9, pp. 635-643, Sept. 1981.

[3] T. Sasao, "An application of multiple-valued logic to a design of masterslice gate array LSI", Proceedings of the 12th International Symposium on Multiple-Valued Logic, May 1982.

[4] M. Davio, J.P. Deschamps and A. Thayse, *Discrete and Switching Functions*, Gerge Publishing Co. and McGraw-Hill, New York, 1978.

[5] S.J. Hong, R.G Cain and D.L. Ostapko, "MINI: A heuristic approach for logic minimization", IBM Res. Develop., Vol. 18, pp. 443-458, Sept. 1974.

[6] R.K. Brayton et al., "A comparison of logic minimization strategies using ESPRESSO: An APL Program package for partitioned logic minimization," Proc. 1982 International Symposium on Circuits and Systems, pp. 42-48. May 1982.

[7] R.J. Nelson, "Simplest normal truth function", J. Symbolic Logic, Vol. 20, pp. 105-108, June 1954.

[8] T. Sasao and H. Terada, "Multiple-valued logic and the design of programmable logic arrays with decoders", Proc. 9th International Symposium on Multiple-valued Logic, May 1979.

[9] S.J. Hong and D.L. Ostapko, "On complementation of Boolean functions", IEEE Trans. on Comput., Vol. C-21, p. 1072, 1972.

[10] D.L. Dietmeyer, *Logic Design of Digital Systems*, (second edition), Allyn and Bacon, Inc., Boston, 1978.

[11] S.Y.H. Su and P.T. Cheung, "Computer simplification of multi-valued switching functions", in *Computer Science and Multiple-Valued Logic*, North-Holland, pp. 189 ~ 220, 1977.

[12] T. Sasao et al., "A fast complementation algorithm for sum-of-products expressions", (in Japanese) Technical Group on Automata and Languages, IECE Japan, Jan. 22, 1981.

[13] R.K. Brayton et al., "Fast recursive Boolean function manipulation", Proc. 1982 International Symposium on Circuit and Systems, pp. 58-62, May 1982.

Appendix

As to the maximum number of the prime implicants of binary functions, the following are known.

Lemma A.1[8]: Let $\mu(n, p)$ be the maximum number of prime implicants of binary functions $P^n \to B$ where $P = \{0,1,...,p\}$. Define $t = 2^p - 1$ and $m = \dfrac{n}{2^p - 1}$. Then

$$(n!)/(m!)^t \leq \mu(n, p)$$

For example for $n=15$ and $p=4$, we have $\mu(p, n) \geq 15! \approx 1.3 \times 10^{12}$.

Theorem A.2[8]: For fixed p, there exists a positive constant K such that

$$K \cdot (t^n / n^{(1-t)/2}) \leq \mu(n, p)$$

where $t = 2^p - 1$.

As to the average number of the prime implicants of binary function, we have the following:

Theorem A.3: Let f be a binary function

$$f: \quad P_i \to B, \quad \text{where} \quad P_i = \{0,1,...,p_i-1\} \quad \text{and}$$

$B = \{0,1\}$, $u = |f^{-1}(1)|$ is a __weight__ of f. The average number of the prime implicants of f with weight u is given by the following:

$$G_p(n,u) =$$

$$\frac{1}{F^{(u)}} \sum_s C^s \sum_{t=0}^{\eta(p,s)} (-1)^t \cdot \sum_t \lambda(p,s,t) \cdot \binom{w - w(t,s)}{u - w(t,s)}$$

where $p = (p_1, p_2, ..., p_n)$, $s = (s_1, s_2, ..., s_n)$, $s \leq p$.

$$C^s = \prod_{i=1}^{n} \binom{p_i}{s_i}, \quad F^{(u)} = \binom{w}{u}, \quad W = \prod_{i=1}^{n} p_i.$$

$\eta(p,s) = \sum_{i=1}^{n} (p_i - s_i)$, $t = (t_1, t_2, ..., t_n)$ is a partition of t, and

$t_i \leq p_i - s_i$,

$$\lambda(p,s,t) = \prod_{i=1}^{n} \binom{p_i - s_i}{t_i}, \quad w(t,s) = \xi(s)\left(1 + \sum_{i=1}^{n} \frac{t_i}{s_i}\right)$$

and $\xi(s) = \prod_{i=1}^{n} s_i$.

(proof.)   Omitted.

Theorem A.4: Average number of the prime implicants of p-valued input binary functions is given by the following:

$$G_p(n) = \sum_k C^k \cdot 2^{-w(k)} \cdot \prod_{i=1}^{p-1} (1 - 2^{-w(k)/i})^{a_i},$$

where $k = (k_1, k_2, ..., k_p)$ is a partition of n.

$$w(k) = \prod_{i=1}^{n} (i)^{k_i}, \quad C^{(k)} = (n!) \prod_{i=1}^{p} \frac{1}{k_i!} \binom{p}{i}^{k_i}, \text{ and } a_i = k_i(p-i).$$

(proof.)   Omitted.

For example, for $n=15$ and $p=4$, we have $G_p(n) \cong 7 \times 10^9$.

The algorithm in section III will generate at most $1/2 \, p^n$ products. For example, for $n=15$ and $p=4$, $1/2 \, p^n \simeq 5 \times 10^8$. This shows that the algorithm generates at least 14 times less products than the elementary one. Table A.1 compares $G_p(n)$ and $1/2 \, p^n$ for $p=2$ and $p=4$.

Table A.1 Comparison with

$G_p(n)$ and $1/2 \, p^n$

n	6	8	10	12	14
$G_2(n)$	24	118	585	2902	14225
$2^n/2$	32	128	512	2048	8192

n	3	4	5	6	7
$G_4(n)$	24	136	758	4095	21565
$4^n/2$	32	128	512	2048	8192

THE SIMPLIFICATION OF MULTIPLE-VALUED SYMMETRIC FUNCTIONS

Jon C. Muzio

University of Victoria
Victoria, B.C.

D. M. Miller

University of Manitoba
Winnipeg, Manitoba

G. Epstein

Indiana University
Bloomington, Indiana

Abstract

A method is given for the synthesis of multiple-valued symmetric function. In an earlier paper a canonical form was derived for the expression of each decisive multiple-valued fundamental symmetric function as a product of certain input terms based on the simple symmetric functions. An algorithm is given for the derivation of maximal product terms which may be used in a representation for a decisive symmetric function. The algorithm is extended to non-decisive symmetric functions and some samples given, in particular it is shown that the algorithm leads to an efficient realization for a ternary full adder.

1. Introduction

This paper is concerned with the synthesis of switching circuits for symmetric and partially-symmetric many-valued functions. It is developed from initial work in [5]. In [7] a canonical form was derived for the expression of each decisive many-valued symmetric function as a product of certain input terms. In this paper our focus of concern is the subclass of decisive symmetric functions as sums of these conjunctions. We discuss the optimization of these results and present an algorithm for the identification of optimal two-level expressions. The results are extended to the synthesis of arbitrary multiple-valued symmetric functions and the example of a ternary full adder is used to illustrate the efficiency of the resulting realization. The question of the allocation of possible "don't cares" and how this may be done in an optimal fashion has been described in [8]. The work reported herein ties together the foundations from [7,8] to give a general method for the synthesis of multiple-valued switching functions.

Our concern is with functions of n variables $x_1,\ldots,x_n$ defined over the finite set $E_r = \{0,1,\ldots,r-1\}$. Within Post algebraic structures these constants would be denoted by e_i, $i = 0,1,\ldots,r-1$ (see [4]). The results deduced herein are valid in general Post algebras because of the normal form theorem and are not limited to Post chains. However our results below are presented for the linearly ordered Post algebras, without loss of generality. X is used to denote $\{x_1,\ldots,x_n\}$ and a function $f(x_1,\ldots,x_n)$ is denoted by $f(X)$ or, when there is no ambiguity, just by f. A function $f(X)$ is _symmetric in the variables_ x_i _and_ x_j if $f(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n)$ $= f(x_1,\ldots,x_j,\ldots,x_i,\ldots,x_n)$. A function is _symmetric_ if it is symmetric in all pairs of its variables (such a function is termed totally symmetric by some authors). If a function is symmetric in at least one pair of variables but not all pairs it is _partially symmetric_.

A _decisive function_ is one which assumes only the values 0 or $r-1$. A _fundamental symmetric function_ within this paper is a decisive symmetric function for which there exists an r-tuple $\{\alpha_0,\ldots,\alpha_{r-1}\}$ such that the function takes the value $r-1$ if, and only if, for each i, $1 \le i \le r-1$, α_i of its arguments assume the value i. In all other cases it will obviously assume the value 0. Clearly $\sum_{i=0}^{r-1} \alpha_i = n$. Fundamental symmetric functions will be denoted by $f_{\alpha_0\alpha_1\cdots\alpha_{r-1}}(X)$ or $f_{\alpha_0\alpha_1\cdots\alpha_{r-1}}$.

Throughout this paper we use a particular set of n basic functions as building blocks. These are the _simple symmetric functions_, denoted by τ_i and defined to be the sum of all possible products of i variables. Hence

$$\tau_1 = x_1 + x_2 + \ldots + x_n$$

$$\tau_2 = x_1 x_2 + x_1 x_3 + \ldots + x_{n-1} x_n$$

$$\vdots$$

$$\tau_n = x_1 x_2 \cdots x_n$$

Here $x + y = $ l.u.b. (x,y) and $xy = $ g.l.b. (x,y). Hence for the linear case $x + y = \max (x,y)$ and $xy = \min (x,y)$.

A number of obvious properties of the simple symmetric functions are noted here. Clearly τ_i takes the value $a(1 \le i \le n; 0 \le a \le r-1)$ if, and only if, at least i variables in X have values greater than $a-1$ and at most $i-1$ variables in X have values greater than a. It follows that $\tau_j \ge \tau_k$ if $1 \le j \le k \le n$. Consequently $\tau_i + \tau_j = \tau_k$, $k = \min(i,j)$ and $\tau_i \tau_j = \tau_k$, $k = \max(i,j)$. As a result sums and products of simple symmetric functions never appear, since they can always be simplified.

2. The Representation of Fundamental Symmetric Functions

Before stating the representation theorem proved in [7] we introduce the unary operators of [6]. Here these are defined by

$$C_i(x) = \begin{cases} r-1 & \text{if } x = i \\ 0 & \text{if } x \ne i \end{cases}$$

The representation theorem below is expressed as a product of terms of the form $C_i(\tau_j)$. For convenience of exposition we denote $C_i(\tau_j)$ by C_{ij}. These C_{ij} give us a useful measure of the numbers of variables assuming certain values; in particular $C_{ij} = r-1$ if, and only if, at least j variables in X have values greater than $i-1$ and at most $j-1$ variables have values greater than i. Consequently if $i > j$ and $k > \ell$ then $C_{ki}C_{\ell j} = 0$ since $\tau_i \le \tau_j$.

We also can make the use of $C_0(C_i(x))$ which is such that

$$C_0(C_i(x)) = \begin{cases} 0 & \text{if } x = i \\ r-1 & \text{if } x \ne i. \end{cases}$$

$C_0(C_{ij})$ will be denoted by $\overline{C_{ij}}$.

It is also useful to have available the unary operators D_i $(0 \le i \le r-1)$ introduced in [4] and defined by

$$D_i(x) = \sum_{j=1}^{r-1} C_j(x)$$

for each i, $0 \le i \le r-1$.

Clearly $D_0(x) = r-1$.

We will denote $D_i(\tau_j)$ by D_{ij} and $C_0(D_{ij})$ by $\overline{D_{ij}}$. Our primary use of D_{ij} is to replace sums of C_{ij}'s by shorter expressions using the result that

$$\sum_{i=k}^{\ell} C_{ij} = D_{kj} \overline{D_{\ell+1,j}}$$

In some contexts one of the pair of D's can be dropped.

Consider a fundamental symmetric function $f_{\alpha_0 \cdots \alpha_{r-1}}$. Define an $(r+1)$-tuple $\{\beta_0, \ldots, \beta_r\}$ by

$$\beta_r = 0$$

$$\beta_i = \beta_{i+1} + \alpha_i \quad \text{for each } i, \quad 0 \le i \le r-1.$$

Theorem 2.1. [7] A fundamental symmetric function can be expressed in the form

$$f_{\alpha_0 \cdots \alpha_{r-1}} = \prod_{\substack{i=0 \\ \alpha_i \ne 0}}^{r-1} C_{i\beta_i} C_{i\beta_{i+1}+1}$$

The expression resulting from the theorem can, in three particular situations be reduced. First if $\alpha_i = 1$ then $\beta_i = \beta_{i+1}+1$ and $C_{i\beta_i} C_{i\beta_{i+1}+1}$ reduces to $C_{i\beta_i}$. At the end of the range if $\alpha_0 \ne 0$ then $C_{0\beta_0}$ is implied by $C_{0\beta_{i+1}}$ and is redundant. Similarly if $\alpha_{r-1} \ne 0$, C_{r-1,β_r+1} is redundant.

It is straightforward to extend Shannon's result for the two-valued case, namely any decisive symmetric function can be expressed as a sum of certain fundamental symmetric functions (each of which is decisive), giving a representation of the decisive symmetric function $f(X)$ in the form

$$f(X) = f_1(X) + f_2(X) + \ldots + f_p(X) \tag{1}$$

where each $f_i(X)$ $(1 \le i \le p)$ is a fundamental symmetric function. The argument which yields this result in the 2-valued case requires only slight modification to give the general result. The actual functions required are easily extracted from a defining table for $f(X)$. We can observe that (1) together with theorem 2.1 could be used to deduce an initial representation for the synthesis of a decisive symmetric function.

In [7] we introduced a diagrammatic approach enabling the result of theorem 2.1 to be read directly off the diagram. We give a brief description of the diagrams here, but refer the reader to [7] for a more formal description.

Any fundamental symmetric function may be represented by a two-dimensional step function. The x-axis has the value set as coordinates, viz $0, 1, \ldots, r-1$ and the y-axis has the number of variables $0, 1, \ldots, n$. For clarity the y-axis is normally drawn to the left of the 0 on the x-axis. Each point on the step function is defined by the number of variables required to assume values less than or equal to the x-coordinate in order for the function to assume the value $r-1$. This is

illustrated in fig. 1 for the function f_{2032}. This function has 7 variables and takes values from $\{0,1,2,3\}$. It is represented by the step function in fig. 1. The x-axis is labelled $0,1,2,3$ and the y-axis by the variables. However the values taken by the β_i defined above by $\beta_r = 0, \beta_i = \beta_{i+1} + \alpha_i$ may be read as a reverse labelling of the y-axis as shown in fig. 1 by the column labelled β. In this example $\beta_4 = 0, \beta_3 = 2, \beta_2 = 5, \beta_1 = 5$, and $B_0 = 7$. To deduce its canonical representation it is only necessary to define sufficient C_{ij} segments to uniquely define its step function. This is illustrated in fig. 2 from which we can write down $f_{2032} = C_{06}C_{25}C_{23}C_{32}$. It is clear from fig. 2 that these four vertical segments uniquely define the desired step function. Note that none of them is redundant.

We can also use the diagrams to directly read off representations for decisive symmetric functions. It is only necessary to give a unique representation of the particular step functions which are illustrated.

For example $f(X) = f_{2032}(X) + f_{2122}(X)$, which is illustrated in fig. 3 can be defined by

$$f(X) = C_{06}C_{23}C_{32}C_{24}\overline{C_{05}},\ \text{as shown in fig. 4.}$$

For more details see [7].

3. The Algorithm for Decisive Symmetric Functions

We present a method for the synthesis of arbitrary symmetric decisive functions. It has some similarity to 2-level minimization of Boolean functions where prime implicants are first derived and a cover found in the second stage. In the first stage all maximal product terms are derived and the second stage chooses and modifies these terms to yield a representation of the desired function. Speed-up techniques could easily be applied to the algorithm.

The aim of the algorithm is to start with some representation of the required decisive symmetric function as a sum of fundamental symmetric functions and develop all the maximal product terms which result from combinations of these fundamental symmetric functions.

For the development of the method we introduce a notation for describing the step functions and superimposed step functions.

A fundamental symmetric function will be described by an n-tuple of values $z_n - z_{n-1} - .. - z_1$ where z_1 is the value assumed by τ_i when the function takes the value $r-1$. This n-tuple is just a representation of the step function. For example f_{2032} the function of 7 variables over $\{0,1,2,3\}$ illustrated in fig. 1, is denoted by $0-0-2-2-2-3-3$. These values can be read directly from fig. 1, summarizing the

information that for f to take the value 3 we require $\tau_7 = 0, \tau_6 = 0, \tau_5 = 2, \tau_4 = 2, \tau_3 = 2, \tau_2 = 3, \tau_1 = 3$. The n-tuple also follows directly from the subscripts in f_{2032} indicating that the n-tuple consists of two 0's followed by zero 1's, three 2's and two 3's.

From the initial list of n-tuples representing fundamental symmetric functions we develop n-tuples of sets $Z_n, .., Z_1$ where each Z_i is a set of values which will lead to possible product terms. A fundamental symmetric function $z_n - z_{n-1} - ... - z_1$ is covered by a product term $Z_n - Z_{n-1} - ... - Z_1$ if, and only if, $z_p \in Z_p$ for all p, $1 \leq p \leq n$ and $z_{p+1} \leq z_p$ for each p, $1 \leq p \leq n-1$. These n-tuples of sets of values correspond directly to the superimposed step function diagrams. Consider figure 5. The 7-tuple of sets of values for this diagram is

$$\{0\},\ \{0\},\ \{0,1,2,3\},\ \{1,2,3,4,5\},\ \{5\},\ \{5,6\},\ \{6\}.$$

Since the full set notation used above for the n-tuples of sets becomes cumbersome in practice it will be abbreviated in the examples. For fig. 5 the representation will be written $0 - 0 - 0123 - 12345 - 5 - 56 - 6$ (since none of our examples will use values exceeding 9 no confusion will arise).

Notice that these sets of values give the vertical segments in the diagram, Z_i giving the values for the vertical segments between $\beta = i$ and $\beta = i-1$. For example $Z_5 = 0123$ corresponds to there being vertical segments between $\beta = 5$ and $\beta = 4$ at values 0, 1, 2, and 3. Likewise $Z_3 = 5$ indicates a single vertical segment between $\beta = 3$ and $\beta = 2$ at value 5. According to our definition above, this term will, for example, cover the fundamental symmetric functions $0 - 0 - 0 - 5 - 5 - 5 - 6$ and $0 - 0 - 2 - 4 - 5 - 6 - 6$ these two being illustrated in fig. 6(a) and (b). The second part of the covering definition, that $z_{p+1} \leq z_p$ for each p, $1 \leq p \leq n-1$ is to ensure that the covered terms represent genuine fundamental symmetric functions and consequently must be non-decreasing.

The algorithm proceeds in n stages. It starts with the list of fundamental symmetric functions whose sum is the desired decisive symmetric function. Each stage yields a new list, the final list consisting of maximal subsets of fundamental symmetric functions. These subsets are termed maximal since it is impossible to add any additional fundamental symmetric functions to them and still represent the result by a single product term.

At each stage terms are combined under certain conditions. Terms are checked if they are covered by a term in the new list. As terms are added to the new list any terms they cover on this new list are deleted. Similarly the term is not added to the new list if it is equal to or covered by a term already on the new list. After all possible new terms

113

are generated all unchecked terms from the old list are appended to the new list. For an r-valued function no more than r terms are ever combined together.

List 1 is the starting list of fundamental symmetric functions and list n+1 is the final list.

The construction of the new lists:

Given list i. To construct list i + 1 (i ≤ i ≤ n). Consider m terms from list i

$$
\left.
\begin{array}{cccc}
Z_{1n} & Z_{1,n-1} & \cdots & Z_{11} \\
Z_{2n} & Z_{2,n-1} & \cdots & Z_{21} \\
\cdot & \cdot & & \cdot \\
\cdot & \cdot & & \cdot \\
\cdot & \cdot & & \cdot \\
Z_{mn} & Z_{m,n-1} & \cdots & Z_{m1}
\end{array}
\right\}
\qquad (2)
$$

This set of m terms is "eligible for combination" if, and only if, for each p (i+1 ≤ p ≤ n)

$$Z_{1p} = Z_{2p} = \cdots = Z_{mp} .$$

This condition is that a set of eligible terms must be identical for columns n, n-1, ..., i+1.

If the set of terms is eligible for combination a new term is formed:

$$W_n W_{n-1} \cdots W_1 .$$

The columns n, n-1, ..., i+1 will be identical to the corresponding columns of (2);

i.e. for each p, i+1 ≤ p ≤ n .

$$W_p = Z_{1p}$$

From the ith column we set $W_i = \bigcup_{j=1}^{m} Z_{ji}$.

The remaining $W_p (1 ≤ p ≤ i-1)$ are slightly more difficult to construct since it is essential to ensure that excluded fundamental symmetric functions are not introduced. Further there are "don't care" possibilities to be incorporated. For example the two terms 0 - 1 - 12 and 0 - 2 - 2 can be combined to 0 - 12 - 12 which, in theory, covers 0 - 1 - 12, 0 - 2 - 2, and 0 - 2 - 1. The latter, however, is an invalid step function and consequently plays the role of a "don't care" function.

Let μ_{jp} = min (Z_{jp}) for each p,j(2 ≤ p ≤ i; 1 ≤ j ≤ m) .

(Note that the columns being used here are i, i-1, .., 2, the minimal entries in the sets in column p will be used in the construction of W_{p-1}).

Let $\mu_p = \min\limits_{1 \le j \le m} (\mu_{jp})$ and we augment each of the Z_{jp} as follows:

For each p,j(1 ≤ p ≤ i-1; 1 ≤ j ≤ m)

$$Y_{jp} = \{y : y \in Z_{jp} \text{ or } \mu_{p+1} \le y \le \mu_{j,p+1}\} .$$

Finally we are in a position to define

$$W_p = \bigcap_{j=1}^{m} Y_{jp}$$

for each p, 1 ≤ p ≤ i-1.

Any $w \in W_{p+1}$ such that $w > \max(W_p)$, 1 ≤ p ≤ i-2, is discarded (it can only cover decreasing terms). Similarly any $w \in W_p$ such that $w < \min(W_{p+1})$, 1 ≤ p ≤ i-1 is dicarded. These two checks are performed successively for $W_{i-1}, W_{i-2}, \ldots, W_1$. Further if at this stage any $W_p (1 ≤ p ≤ i-1)$ is empty the entire term is void and is discarded. Otherwise it is added to list i+1.

Example 3.1. An example will clarify the procedure. This is a 6-valued example with a function of 4 variables. Consider the following three terms from list 3. They are illustrated in fig. 7.

```
0 - 1 - 1 2 3 -   3 4
0 - 2 -   2 3 -   4 5
0 - 3 -     3 - 3 4 5
```

We have i = 3 and

$$Z_{14} = \{0\} \ Z_{13} = \{1\} \ Z_{12} = \{1,2,3\} \ Z_{11} = \{3,4\}$$

$$Z_{24} = \{0\} \ Z_{23} = \{2\} \ Z_{22} = \{2,3\} \quad Z_{21} = \{4,5\}$$

$$Z_{34} = \{0\} \ Z_{33} = \{3\} \ Z_{32} = \{3\} \quad Z_{31} = \{3,4,5\}$$

Since $Z_{14} = Z_{24} = Z_{34}$ these terms are eligible for combination and $W_4 = \{0\}$.

$$W_3 = \bigcup_{j=1}^{3} Z_{j3} = \{1,2,3\}$$

For W_2 and W_3 we have

$\mu_{13} = 1$	$\mu_{12} = 1$
$\mu_{23} = 2$	$\mu_{22} = 2$
$\mu_{33} = 3$	$\mu_{32} = 3$

so

$$\mu_3 = 1 \qquad \mu_2 = 1$$

Hence

$$Y_{12} = \{y : y \in Z_{12} \text{ or } \mu_3 \le y < \mu_{13}\}$$

$$= \{1,2,3\}$$

Similarly

$$Y_{22} = \{1,2,3\}$$

$$Y_{32} = \{1,2,3\}$$

so $W_2 = \{1,2,3\}$.

For W_1 we have

$$Y_{11} = \{3,4\}$$

$$Y_{21} = \{1,4,5\}$$

$$Y_{31} = \{1,2,3,4,5\} \qquad \text{so } W_1 = \{4\}.$$

114

The resulting combined term to be entered on list 4 is

$$0 - 123 - 123 - 4 .$$

This term is illustrated in fig. 8. Examination reveals that every step function of fig. 8 is included in one of the diagrams of fig. 7. Notice that none of the three original terms is covered by the new term, so none of them will be checked. The extra values introduced into the sets Y_{jp} correspond to the introduction of certain vertical segments, the dashed lines in fig. 7, which are don't care conditions.

The algorithm can be written in a more convenient form for hand execution. The division between the Z_{jp} and the extra values introduced into Y_{jp} is indicated by a slash mark (/). The above example appears as below:

$$0 - 1 - /123 - /34$$
$$0 - 2 - 1/23 - 1/45$$
$$0 - 3 - 12/3 - 12/345$$

giving $0 - 123 - 123 - 4$.

Prior to looking at the example in rather more detail we note a few points regarding the application of this algorithm.

When m terms of a particular list do not combine it is still possible that certain subsets will combine. Further even if the m rows do combine some subset may also produce a useful term. If m terms do combine then any subset will combine and the term generated by the subset is not necessarily covered by the term generated by the m rows.

This is illustrated using the above example, the three terms illustrated in fig. 7 combining to give the term in fig. 8. However we also can combine the terms in pairs, namely

$$0 - 1 - /123 - /34$$
$$0 - 2 - 1/23 - 1/45$$
give $$0 - 12 - 123 - 4$$

$$0 - 1 - /123 - /34$$
$$0 - 3 - 12/3 - 12/345$$
give $$0 - 13 - 123 - 34$$

$$0 - 2 - /23 - /45$$
$$0 - 3 - 2/3 - 2/345$$
give $$0 - 23 - 23 - 45$$

These three terms are illustrated in fig. 9 and we note that only the first is covered by the term generated by all three.

In general all subsets must be tried. The procedure adopted is as follows:

Select m rows that are eligible for combination, that is

$$Z_{1p} = Z_{2p} = \ldots = Z_{mp} \text{ for each } p(i + 1 \leq p \leq n)$$

and, to ensure the new row will be distinct from all

m rows, $Z_{ki} \neq Z_{\ell i}$ for some k, $\ell(1 \leq k, \ell \leq m)$.

The most efficient approach appears to be to attempt to combine subsets in increasing order of size, that is pairs then triplets etc. This order is used since the failure of a certain subset of rows to combine indicates that all larger subsets containing this subset will also fail and need not be attempted. As terms are generated for the new list they must be examined to determine if they cover or are covered by a previously generated term. Covered terms are deleted.

Example 3.2 The same 4-variable six-valued example of example 3.1 but with all the lists. Covered terms have been omitted from later lists.

The required function is the sum of the following fundamental symmetric functions:

$$f = f_{120100} + f_{120010} + f_{111100} + f_{111010} + f_{110200}$$
$$+ f_{110110} + f_{102010} + f_{102001} + f_{101110} + f_{101101}$$
$$+ f_{100300} + f_{100210} + f_{100201}$$

These functions are listed (in the same order) in list 1.

List 1

```
0 - 1 - 1 - 3✓
0 - 1 - 1 - 4✓
0 - 1 - 2 - 3✓
0 - 1 - 2 - 4✓
0 - 1 - 3 - 3✓
0 - 1 - 3 - 4✓
0 - 2 - 2 - 4✓
0 - 2 - 2 - 5✓
0 - 2 - 3 - 4✓
0 - 2 - 3 - 5✓
0 - 3 - 3 - 3✓
0 - 3 - 3 - 4✓
0 - 3 - 3 - 5✓
```

List 2

```
0 - 1 - 1 - 34✓
0 - 1 - 2 - 34✓
0 - 1 - 3 - 34✓
0 - 2 - 2 - 45✓
0 - 2 - 3 - 45✓
0 - 3 - 3 - 345
```

List 3

```
0 - 1 - 123 - 34✓
0 - 2 - 23 - 45✓
0 - 3 -   3 - 345
```

List 4

```
0 - 13 - 123 - 34
0 - 23 -  23 - 45
0 -123 - 123 - 4
0 - 3 -   3 - 345
```

List 5 is identical to List 4.

Consequently the conclusion of this portion of the procedure is the four maximal product terms given above.

A slight adjustment to the practical application of the algorithm is often made in order to make it a little easier to use. It will be recalled that when combining m rows we defined

$$Y_{jp} + \{y : y \in Z_{jp} \quad \text{or} \quad \mu_{p+1} \leq y < \mu_{j,p+1}\}$$

The problem with this is that μ_{p+1} varies when we consider just some subsets of the rows and it is necessary to recalculate all the Y_{jp}.

For example

$$0 - 1 - /123 - /34$$
$$0 - 2 - 1/23 - 1/45 \qquad (3)$$
$$0 - 3 - 12/3 - 12/345 \qquad (4)$$
give $$0 - 123 - 123 - 4$$

but $\qquad$ $0 - 2 - /23 - /45 \qquad (5)$

$\qquad\qquad 0 - 3 - 2/3 - 2/345 \qquad (6)$

give $\qquad 0 - 23 - 23 - 45 \qquad (7)$

In practice it is easier to use the μ_{p+1} value for the maximum number of eligible rows when constructing the Y_{jp} and then when subsets are combined delete all values in W_p which are less than μ_{p+1} calculated for just the subset.

If m is the maximum number of eligible rows, with $\mu_p = \min\limits_{1\leq j\leq m}(\mu_{jp})$ and we are combining some sub-

set A of terms $\{1,2,..,m\}$ then

$Y_{jp} = \{y : y \in Z_{jp}$ or $\mu_{p+1} \leq y < \mu_{j,p+1}\}$ but for

subset A

$W_p = \{w : w \in \bigcup\limits_{j=1}^{m} Y_{jp}$ and $w \geq \min\limits_{j\in A}(\mu_{j,p+1})\}$

The effect of this is that we can still use (3) and (4) directly rather than (5) and (6) to deduce (7) above, i.e.,

$\qquad\qquad 0 - 2 - 1/23 - 1/45$

$\qquad\qquad 0 - 3 - 12/3 - 12/345$

would give $\quad 0 - 23 - 123 - 145$

but the two 1's are deleted by the $w \geq \min\limits_{j\in A}(\mu_{J,p+1})$ condition.

4. The Algorithm for Arbitrary Symmetric Functions

As was mentioned in section 3 an arbitrary symmetric function $f(Z)$ may always be expressed in the form

$f(Z) = 1g_1(Z) + 2g_2(Z)+..+(r-2)g_{r-2}(Z) + g_{r-1}(Z)$

where each $g_i(Z)$, $1 \leq i \leq r-1$ is a decisive symmetric function. With a number of modifications the algorithm of the last section can be applied. Initially we should note that all the cases included in $g_i(Z)$ (some i, $2 \leq i \leq r-1$) become don't cares for all $g_j(Z)$, $1 \leq j < i$. Consequently any fundamental symmetric function contributing to the realization of $g_i(Z)$ becomes a don't care function for all $g_j(Z)$, $1 \leq j < i$. They may be included or excluded solely to optimize the representation of each $g_j(Z)$. We shall attack this problem in a manner analogous to that used for don't care minterms in prime implicant techniques.

The Modified Algorithm

Each term in the lists in the algorithm has an appended value attached to it. In the initial list of fundamental symmetric functions v_p is the value assumed by the symmetric function when the corresponding fundamental symmetric function assumes the value $r-1$. When m terms are combined the new term $W_n \ldots W_1$ has $\min\limits_{1\leq j\leq m}(v_j)$ as its associated value. When checking for covered terms the associated value must also be checked. An example will clarify the procedure.

Example 4.1 To illustrate the algorithm we consider

the design of a ternary full adder, a circuit which is of practical importance and has been considered by many authors. (see [3,10]).

A full adder has three three-valued variables as inputs (to allow for two inputs and a carry-in) and produces two outputs, the sum f_s and the carry f_c. To maintain the symmetry of the unit we allow the carry-in to assume the value 2 and, as a result, the carry-out can also assume the value 2 in one situation.

The function is defined by the lists in table 1 - note the value taken by the function for each term in the list is appended to the term. Detail discussion is limited to the sum output f_s and list 1 for this is taken from table 1. In terms of the fundamental symmetric functions the required sum and carry functions are:

$f_s = 1(f_{210} + f_{102} + f_{021}) + f_{201} + f_{120} + f_{012}$

$f_c = 1(f_{012} + f_{102} + f_{021} + f_{111} + f_{030}) + f_{003}$

Table 1

For f_s	For f_c
0 - 0 - 1 : 1	1 - 2 - 2 : 1
0 - 2 - 2 : 1	0 - 2 - 2 : 1
1 - 1 - 2 : 1	0 - 1 - 2 : 1
0 - 0 - 2 : 2	1 - 1 - 2 : 1
0 - 1 - 1 : 2	1 - 1 - 1 : 1
1 - 2 - 2 : 2	2 - 2 - 2 : 2

The six fundamental symmetric functions for f_s are illustrated in fig. 10.

The application of the algorithm is given in table 2. On list 1 rows are only eligible for combination if they agree in the first two columns, so the only eligible subsets are $0 - 0 - 1 : 1$ and $0 - 0 - 2 : 2$, the combined term being the first on list 2 and covering just $0 - 0 - 1 : 1$. The remaining uncovered terms from list 1 are appended to list 2.

On list 2 the terms which are eligible for combination are
$[0 - 0 - 12 : 1, 0 - 2 - 2 : 1, 0 - 1 - 1 : 2]$,
$[0 - 2 - 2 : 1, 0 - 0 - 2 : 2, 0 - 1 - 1 : 2]$, and
$[1 - 1 - 2 : 1, 1 - 2 - 2 : 2]$. We consider the three cases in detail.

Table 2

List 1	List 2
0 - 0 - 1 : 1✓	0 - 0 - 12 : 1✓
0 - 2 - 2 : 1	0 - 2 - 2 : 1✓
1 - 1 - 2 : 1	1 - 1 - 2 : 1✓
0 - 0 - 2 : 2	0 - 0 - 2 : 2
0 - 1 - 1 : 2	0 - 1 - 1 : 2
1 - 2 - 2 : 2	1 - 2 - 2 : 2

List 3	List 4
0 - 02 - 12 : 1✓	01 - 02 - 12 : 1
0 - 01 - 1 : 1	0 - 01 - 1 : 1
1 - 12 - 2 : 1	1 - 12 - 2 : 1
0 - 0 - 2 : 2	0 - 0 - 2 : 2
0 - 1 - 1 : 2	0 - 1 - 1 : 2
1 - 2 - 2 : 2	1 - 2 - 2 : 2

Consider

$$0 - 0 - /12 : 1 \qquad (8)$$
$$0 - 2 - 01/2 : 1 \qquad (9)$$
$$0 - 1 - 0/1 \ : 2 \qquad (10)$$

(8) and (9) combine to give $0 - 02 - 12 : 1$ which covers both (8), (9).

(8) and (10) give $0 - 01 - 1 : 1$.

(9) and (10) give $0 - 12 - 1 : 1$ which reduces back to $0 - 1 - 1 : 1$ which is a degenerate form of (10). Consequently (8), (9), (10) together need not be tried.

From
$$0 - 2 - 01/2 : 1 \qquad (11)$$
$$0 - 0 - /2 \ : 2 \qquad (12)$$
$$0 - 1 - 0/1 : 2 \qquad (13)$$

we deduce

$0 - 02 - 2 : 1$ from (11), (12)

(this is covered by the first term on list 3 and so not included).

$0 - 12 - 1 : 1$ from (11), (13)
This is a degenerate term.

(12) and (13) do not combine.

Finally
$$1 - 1 - /2 \ : 1$$
$$1 - 2 - 1/2 : 2$$
give $\quad 1 -12 - 2 \quad : 1$ which covers the former term.

List 3 results by adding the uncovered rows from list 2. List 4 follows by a similar procedure applied to list 3, list 3 with the Y_{jp} sets being given in Table 3. List 4 consists of the maximal subsets of fundamental symmetric functions. The first three terms are illustrated in fig. 11, the last three appearing as (d), (e), (f) in fig. 10.

Table 3

List 3
$$0 - \ /02 - \ /12 : 1$$
$$0 - \ /01 - \ /1 \ : 1$$
$$1 - 0/12 - 0/2 \ : 1$$
$$0 - \ /0 \ - \ /2 \ : 2$$
$$0 - \ /1 \ - 0/1 \ : 2$$
$$1 - 0/2 \ - 01/2 : 2$$

For this example we can now read off expressions from figs. 10 and 11 for g_1 and g_2 where $f = 1g_1 + g_2$. For g_2 the three relevant terms each represent a fundamental symmetric function and the required expression is just the sum of the three, viz:

$$g_2 = C_{02} \ C_{21} + C_{03} \ C_{12} \ C_{11} + C_{13} \ C_{22}$$

For g_1 the choice of a best representation is rather more difficult since as soon as one term is chosen it becomes a "don't care" for all other terms for g_1. A detailed discussion of an appropriate selection procedure is beyond the scope of this paper and may be found in [10]. Here we content ourselves with giving two possible representations for g_1.

Either $\quad g_1 = C_{13} \ C_{21} + C_{03} \ \overline{C}_{12} \ \overline{C}_{01}$

or $\quad g_1 = C_{13} \ C_{21} + C_{03} \ (C_{11} + C_{22})$.

For f_c a similar process leads to two possible leads to two possible expressions, viz

$$f_c = (C_{13} + C_{22} + C_{03} \ C_{21}) \ 1 + C_{23}$$

or $\quad f_c = (C_{21} \ \overline{C}_{02} + C_{13}) \ 1 + C_{23}$

A realization for the full adder is illustrated in fig. 11 based on gates to realize $+$ and $..$ Such gates have been described by many authors (see, for example [9,10]). In addition to the illustrated circuit, additional hardware is used to generate τ_1, τ_2, and τ_3 together with all the C_{ij} functions.
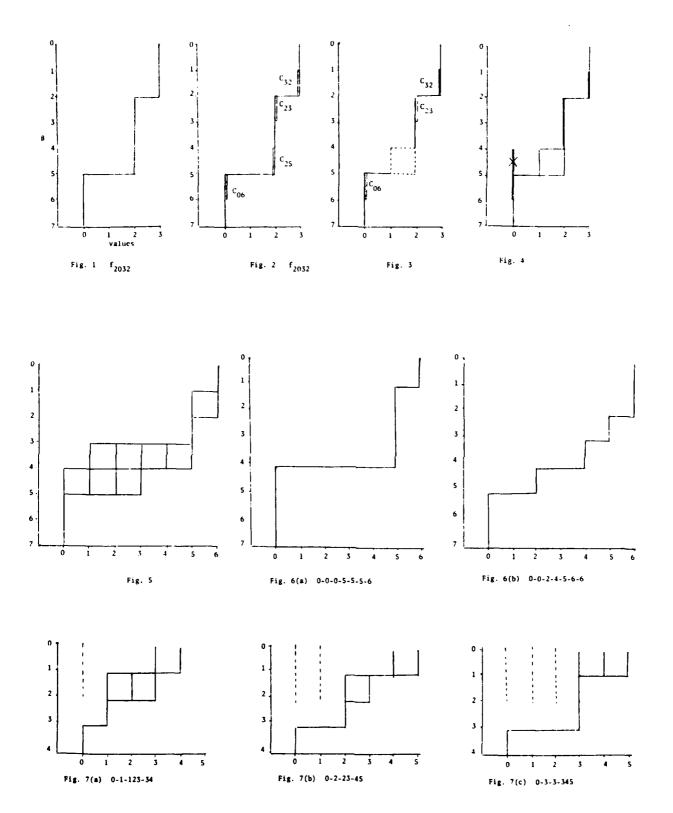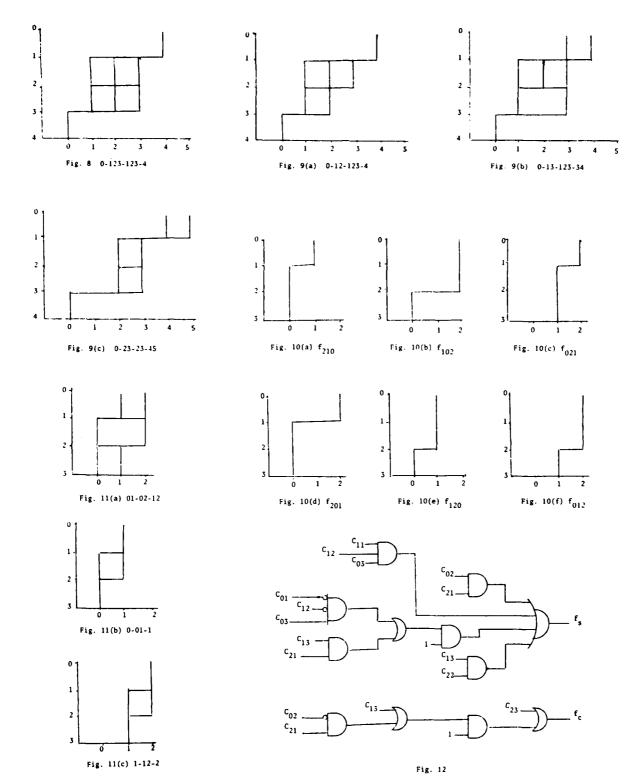
5. Summary

The major result presented here is the algorithm for the synthesis of decisive symmetric functions and its extension to arbitrary symmetric functions. The technique used is a systematic method for the derivation of all possible maximal product terms together with the incorporation of relevant "don't cares".

The attractiveness of our method is that it leads to much simpler expressions for the functions (for example the ternary full adder expressions in [10] contain 27 terms in the sum of products realizations). This leads to simpler implementations of the results.

References

1. Current, K.W. and Mow D.A., Implementing parallel counters with four-valued threshold logic, IEEE Trans. Comput., vol. C-28, 200-204, 1979.

2. Current, K.W., Pipelined binary parallel counters employing latched quaternary logic full adders, IEEE Trans. Comput., vol. C-29, 400-403, 1980.

3. Dao, T.T., Davio, M. and Gossart, C., Complex number arithmetic with odd-valued logic, IEEE Trans. Comput., vol. C-29, 604-610, 1980.

4. Epstein, G., The Lattice theory of Post algebras, Transactions of the American Mathematical Society 95, 300-317, May, 1960.

5. Epstein, G., General synthesis of electronic circuits for symmetric functions, Computer Science Conference Abstracts, Columbus, Ohio, 35, Feb., 1973.

6. Epstein, G. and Horn, A., Chain-based lattices, Pacific J. Math., Vol. 55, 65-84, 1974.

7. Epstein, G., Miller, D.M. and Muzio, J.C., Some preliminary views on the general synthesis of electronic circuits for symmetric and partially symmetric functions, Proc. Seventh International Symposium on Multiple-Valued Logic, 29-34, 1977.

8. Epstein, G., Miller, D.M. and Muzio, J.C., Selecting don't care sets for symmetric n-valued functions: a pictorial approach using matrices, Proc. Tenth International Symposium on Multiple-Valued Logic, June, 1980.

9. McCluskey, E.J., Logic design of multivalued I^2L Logic Circuits, IEEE Trans. Comput., vol. C-28, 546-559, 1979.

10. Mouftah, H.T. and Jordan, I.B., A design technique for an integrable ternary arithmetic unit, Proc. Fifth International Symposium on Multiple-Valued Logic, pp. 359-372, 1975.

117

Fig. 1 f_{2032} Fig. 2 f_{2032} Fig. 3   Fig. 4

Fig. 5   Fig. 6(a)   0-0-0-5-5-5-6   Fig. 6(b)   0-0-2-4-5-6-6

Fig. 7(a)   0-1-123-34   Fig. 7(b)   0-2-23-45   Fig. 7(c)   0-3-3-345

118

Fig. 8 0-123-123-4

Fig. 9(a) 0-12-123-4

Fig. 9(b) 0-13-123-34

Fig. 9(c) 0-23-23-45

Fig. 10(a) f_{210}

Fig. 10(b) f_{102}

Fig. 10(c) f_{021}

Fig. 11(a) 01-02-12

Fig. 10(d) f_{201}

Fig. 10(e) f_{120}

Fig. 10(f) f_{012}

Fig. 11(b) 0-01-1

Fig. 11(c) 1-12-2

Fig. 12

119

Session 3B
Algebra I

SELFDUAL CLASSES AND AUTOMORPHISM GROUPS

by J. Demetrovics, L. Hannák, L. Rónyai

Computer and Automation Institute, Hungarian Academy of
Sciences, Budapest, Hungary

Abstract

In this paper the atuhors investigate some problems connected with selfdual closed classes and permutation groups. A typical problem in this direction is the following. Let G be a permutation group. What is the cardinality of closed classes contained in the centralizer of G? We review some earlier results and discuss the case of "small" permutation groups. Some open problems are formulated, too.

Introduction

Let P_k be the algebra of finitary functions over the set $E_k=\{0,1,\ldots,k-1\}$ and D a closed class in P_k (D is a nonempty set of functions closed under composition, permutation and indentification of variables.) If we define $\nu(D)$ as the number (cardinality) of the closed classes contained in D, we can consider the following problem: what is the value of $\nu(D)$ for a closed class D of a given type.

Many results are known in this direction for "large" subclasses of P_k. For example, Post's classical results [14] imply $\nu(P_2)=\kappa_0$. Janov and Mucnik [7] have given a construction for k>2 which implies $\nu(P_k)=2^{\kappa_0}$. Demetrovics and Hannák [5] have shown that for k>2 and a precomplete class D defined by a partial order, by a relation which is either equivalence or central or h-regular (i.e. D=Pol ρ where ρ is a relation from the listed four types) we have $\nu(D)=2^{\kappa_0}$. The most complete description has been given in the case of a quasi-linear subclass of P_k (see Salomaa [16], Demetrovics-Bagyinszki [4], Szendrei [17], [18], Bagyinszki [1], Lau [8]).

The aim of this paper is to study the function ν for selfdual closed classes.

Denote by S_k the full symmetric group on E_k. We can extend $\pi \in S_k$ to E_k^n by setting $\pi(\underline{x})=(\pi(x_1),\ldots,\pi(x_n))$ for all

$\underline{x}=(x_1,\ldots,x_n)\in E_k^n$.

Definition: The function $f\in P_k$, $f:E_k^n \to E_k$ is said to be <u>selfdual to</u> $\pi \in S_k$ if $\pi(f(\underline{x}))=f(\pi(\underline{x}))$ for all $\underline{x}\in E_k^n$. (i.e. π is an automorphism of the algebra $\langle E_k, f\rangle$.)

Definition: Let $G\le S_k$. The centralizer C(G) of G is defined as $C(G)=\{f; f\in P_k,$ and f is selfdual to each $\pi \in G\}$. If $\pi \in S_k$ has no fixed points and if it can be written as the product of disjoint cycles of the same length then $C(\pi)=C(\langle\pi\rangle)$ is a precomplete class of P_k (Rosenberg [15]).

Define
$$c(H)=\nu(C(H))$$

The following propositions collect some general statements about the function c.

Proposition 1.: Let $H,K\le S_k$ and $\pi \in S_k$. Then

a/ $C(H)= \underset{\xi \in H}{C}(\xi)$

b/ $c(\pi^{-1}H\pi)=c(H)$

c/ if K>H then C(K)<C(H) and $c(K)\le c(H)$.

Proof: a/ Obvious from the definition.

b/ Consider the mapping $\bar{\pi}:P_k \to P_k$, $\bar{\pi}(f)=\pi^{-1}f\pi$ $(f\in P_k)$. It is easy to verify that π is an automorphism of the preiterative Post algebra P_k and that the induced mapping is also an automorphism of $L(P_k)$. By the first statement it is enough to prove that for every $\xi \in S_k$ and $f\in P_k$

$f\in C(\xi)<=>\bar{\pi}(f)\in C(\pi^{-1}\xi\pi)$.

This equivalence can be proved by an easy computation. To prove c/ we use the following construction of Pálfy-Szendrei-Szabó [13]: let the k-ary function f_H be

defined by

$$f_H(a_1,\ldots,a_k)=\begin{cases} a_1 & \text{if there exists } \xi\epsilon H \text{ such} \\ & \text{that } \xi(i)=a_{i+1} \text{ for all} \\ & i\epsilon E_k \\ a_2 & \text{in other cases.} \end{cases}$$

The definition implies immediately that $f_H\epsilon C(\xi)<=> \xi \epsilon H$, hence for $K>H, f_H\epsilon C(H)$ and $f_H\notin C(K)$ hold. This proves c./ □

Earlier results

The values of the function c are known for some fixed permutation groups. Demetrovics and Hannák [6] proved the following statements:

<u>Proposition 2.:</u> Let $k>2$ and let $\xi\epsilon S_k$ be such that $\xi=\xi_1\ldots\xi_s$ where $\xi_1,\ldots,\xi_s$ are disjoint cycles. If there are $1\le i\ne j\le s$ such that the length $(\xi_i)>1$ and length $(\xi_j)|$length (ξ_i) then $c(\xi)=2^{\kappa_0}$. □

<u>Proposition 3.:</u> If $\xi\epsilon S_k$ contains a cycle of length at least five then $c(\xi)=2^{\kappa_0}$. □

<u>Proposition 4.:</u> Let $\xi\epsilon S_k$ be such that $\xi=\xi_1\xi_2$ where ξ_1 and ξ_2 are disjoint cycles. If length $(\xi_1)=2$ length $(\xi_2)=3$ or length $(\xi_1)=3$, length $(\xi_2)=4$ then $c(\xi)=2^{\kappa_0}$ □
The proofs use modified versions of the above mentioned construction of Janov and Mucnik [7].

As a consequence of above three propositions we can obtain the following.

<u>Theorem 1.:</u> (Demetrovics-Hannák [6]).

Let $k\ge 3$ and $\xi\epsilon S_k$. Then $c(\xi)=2^{\kappa_0}$ excluding the cases when either
    a/ $k=3$ and ξ is a 3-cycle
    b/ $k=4$ and ξ is a 4-cycle
In these exceptional cases $c(\xi)\ge\kappa_0$ hold. □

<u>Corollary 1.:</u> Let $k\ge 4$, $G\le S_k$ be a cyclic subgroup. If $c(G)<2^{\kappa_0}$ then $k=|G|=4$ and $c(G)\ge\kappa_0$. □

<u>Remark:</u> Recently Marcenkov [11] proved that in the case a/ $c(\xi)=2^{\kappa_0}$ also holds. In the case b/ the exact value of $c(\xi)$ is unknown.

<u>Problem 1.:</u> Find the exact value $c(G)$ for exceptional cases. (The remark suggests that in these cases $c(G)=2^{\kappa_0}$ holds, too.)
In the above statements we have discussed c for some "small" subgroups of S_k. On the other hand let us consider "large" subgroups of S_k.

Using the terminology of Marczewski [12], Csákány [2] and Marcenkov [9], the elements of $C(S_k)$ and the subclasses of $C(S_k)$ will be called homogenous functions and homogenous closed classes, respectively. The structure of $C(S_2)$ has been described by Post who also showed that $c(S_2)=7$. There are some intresting results about special homogenous functions (e.g. the ternary discriminator, dual discriminator) as well as about the structure of arbitrary homogenous classes.

The structure of $C(S_3)$ has been determined by Csákány [2] and Marcenkov [9]. There are exactly seven nontrivial three element homogenous algebras hence $c(S_3)=8$.

For $k\ge 3$ Csákány and Gavalcová [3] have described all minimal nontrivial homogenous algebras and also have listed $2k-1$ distinct nontrivial homogenous algebras. The exact result in this direction was proved by Marcenkov [10], who showed that the number of nontrivial homogenous algebras is 14 for $k=4$ and $4k-3$ for $k>4$. So $c(S_4)=15$ and $c(S_k)=4k-2$ for $k>4$.

An other intresting result of [10] is $c(S)=\kappa_0$ where S denotes the full symmetric group on $E=\{0,1,2,\ldots,n,\ldots\}$. The proof of this statement is also constructive. Marcenkov have listed all subclasses of $C(S)$.

Knowing the above results, the following natural questions arise:

<u>Problem 2.:</u> Determine the value of $c(A_k)$ where A_k is the alternating group on E_k.

From the above results and using $A_2=\{id\}$ we know $c(A_2)=\kappa_0$ and $c(A_3)=2^{\kappa_0}$.

<u>Problem 3.:</u> For $k\ge 4$ find a subgroup $G\le S_k$ $c(G)=\kappa_0$.

Blocks of groups

In the following we are going the prove an inequality for c.

<u>Definition:</u> Let $H\le S_k$, a proper subset D of E_k is said to be a <u>block</u> of H if for each $\xi\epsilon H$ either $\xi(D)=D$ or $\xi(D)\cap D=\phi$.
If D is a block of H then let $H_D=\{\pi\epsilon S_D: \exists \xi\epsilon H \text{ such that } \xi(x)=\pi(x) \text{ for all } x\epsilon D\}$.

<u>Theorem 2.:</u> Let D be a block of H. Then
$$c(H)\ge c(H_D).$$

The proof of this theorem depend on the following two propositions.

123

<u>Proposition 5.:</u> Let $g: E_k^n \to E_k$ be a partial function and $H \leq S_k$. Suppose that for all $\pi \in H$

$$\pi g(\underline{x}) = g(\pi(\underline{x}))$$

holds whenever $\underline{x}$, $\pi(\underline{x}) \in \text{dom } g$. There exists a full operation $\tilde{g} \in C(H)$ for which

$$\tilde{g}(\underline{x}) = g(\underline{x}) \text{ if } \underline{x} \in \text{dom } g.$$

<u>Proof:</u> Let $D_1, \ldots, D_s$ the orbits of H on E_k^n for which $D_i \cap \text{dom } g \neq \emptyset$, $\underline{x}^i \in D_i \cap \text{dom } g$ and $y_i = g(\underline{x}^i)$ $(1 \leq i \leq s)$. We define the function $\tilde{g}$ as

$$\tilde{g}(\underline{x}) = \begin{cases} \pi(y_i) \text{ if } \underline{x} \in D_i \text{ for some } (1 \leq i \leq s) \text{ and} \\ \pi \in H \text{ an arbitrary permutation for} \\ \quad \text{which } \pi(\underline{x}^i) = \underline{x}. \\ x_1 \text{ in other cases.} \end{cases}$$

It's clear that $\tilde{g}$ is a full operation. We remark that if $\underline{x} \in D_i$ then $\tilde{g}(\underline{x})$ is independent from the choice of π. If $\pi, \xi \in H$, $\pi(\underline{x}^i) = \xi(\underline{x}^i) = \underline{x}$ then $\xi^{-1}\pi$ stabilizes $\underline{x}^i$ and $\underline{x}^i \in \text{dom } g$. This implies that $y_i = g(\underline{x}^i) = \xi^{-1}\pi g(\underline{x}^i) = \xi^{-1}\pi(y_i)$ and $\pi(y_i) = \xi(y_i)$. Now let $\underline{x} \in E_k^n$ and $\xi \in H$. We distinguish two cases:

α, There is an i, $1 \leq i \leq s$ for which $\underline{x} \in D_i$. Then $\pi(\underline{x}^i) = \underline{x}$ for some $\pi \in H$ and using above remark $\tilde{g}(\underline{x}) = \pi(y_i)$ and $\tilde{g}(\xi(\underline{x})) = \xi\pi(y_i)$ hold. Using these observations we obtain $\xi\tilde{g}(\underline{x}) = \tilde{g}(\xi(\underline{x}))$.

β, $\underline{x} \notin D_i$ for every $1 \leq i \leq s$. Then

$$\xi\tilde{g}(\underline{x}) = \xi(x_1) = \tilde{g}(\xi(\underline{x})).$$

In both cases ξ and $\tilde{g}$ commute and $\tilde{g} \in C(H)$. If $\underline{x} \in \text{dom } g$ then $\underline{x} \in D_i$ for some i and $\underline{x} = \pi(\underline{x}^i)$ for an appropriate $\pi \in H$.

$$g(\underline{x}) = g(\pi(\underline{x}^i)) = \pi g(\underline{x}^i) = \pi(y_i) = \tilde{g}(\underline{x})$$

This proves the last assertion. $\square$

<u>Proof of Theorem 2.:</u>
First we prove that for all $f: D^n \to D$, $f \in C(H_D)$ there exists an $\bar{f}$, $\bar{f}: E_k^n \to E_k$ such that $\bar{f} \in C(H)$ and $\bar{f}(\underline{x}) = f(\underline{x})$ for all $\underline{x} \in D^n$. By prop. 5. it is enough to prove that for each $\xi \in H$, $\underline{x}$, $\xi(\underline{x}) \in \text{dom} f$, $\xi(f(\underline{x})) = f(\xi(\underline{x}))$ holds. Since $\xi(\underline{x}) \in D^n$, $D \cap \xi(D) \neq \emptyset$, and $\xi(D) = D$. In this case $\pi = \xi_{|D} \in H_D$ and

$$\xi(f(\underline{x})) = \pi(f(\underline{x})) = f(\pi(\underline{x})) = f(\xi(\underline{x}))$$

so there exists an extension $\bar{f}$ of f with the wanted properties.
Let $K \leq C(H_D)$ be a closed class and let $\bar{K} = \{f; f \in C(H) \text{ and } \exists g \in K \text{ such that}$ $f(\underline{x}) = g(\underline{x}); \underline{x} \in \text{dom} g\}$.

Clearly $\bar{K}$ is a closed class contained in $C(H)$ and if $K, K' \leq C(H_D), K' \neq K$ then $\bar{K}' \neq \bar{K}$. The mapping $K \to \bar{K}$ is injective showing $c(H) \geq c(H_D)$. $\square$

<u>Definition:</u> A group $H \leq S_k$ is called semiregular if only the identity element of H has a fixed point.

<u>Corollary 2.:</u> If $H \leq S_k$ semiregular and not a 2-group then $c(H) = 2^{\aleph_0}$.

<u>Proof:</u> In this case there exists a block D of H such that $|D| = p$, $p \neq 2$ prime and H_D is a cyclic group of order p. By cor.1. $c(H_D) = 2^{\aleph_0}$ and Theorem 2. implies $c(H) = 2^{\aleph_0}$ $\square$

<u>Corollary 3.:</u> Let $p > 2$, p prime, $k > 2$ and $P \leq S_k$ a p-subgroup. Then $c(P) = 2^{\aleph_0}$.

<u>Proof:</u> If $P = \{\text{id}\}$ then the statement is obvious. If $P \neq \{\text{id}\}$ then it has a block D for which $|D| = p$ and P_D is a cyclic group of order p. If $H \leq Z(P)$ and $|H| = p$ then H has an orbit D of length p. This orbit obviously is a block of P and thus it satisfies the requests. Now as in cor.2. we obtain the assertion. $\square$

Acknowledgements

References

[1] J.Bagyinszki, The lattice of closed classes of linear functions defined over a finite ring of square-free order, K.Karl Marx Univ. of Economics Dept. Math. Budapest, vol. 2, 1979.

[2] B.Csákány, Homogenous algebras are functionally complete, Algebra Universalis /to appear/

[3] B.Csákány-T.Gavalcova, Finite homogeneous algebras, Acta Sci.Math./Szeged/ 42 /1980/, 57-65.

[4] J.Demetrovics-J.Bagyinszki, The lattice of linear classes in prime valued logic, Banach Center Publ., Warsaw, PWN, v.8. 1979.

[5] J.Demetrovics-L.Hannák, The cardinality of closed sets in precomplete classes in k-valued logics, Acta Cybernetica,4 /1979/,3, 273-277.

[6] J.Demetrovics-L.Hannák, The number of reducts of a preprimal algebra, Proc. ISMVL'82. 331-335.

[7] Ju.J.Janov-A.A.Mucnik, Existence of k-valued closed classes without finite basis /in Russian/, Dokl. Akad. Nauk SSSR, 127/1959/, 44-46.

[8] D.Lau, Über die Anzahl abgeschlossen Mengen von linearen Funktionen der n-wertigen Logik, Elektron. Inform. Kybernet. 14 /1978/ 567-569.

[9] S.S.Marcenkov, On closed classes of selfdual functions of many-valued logic /in Russian/, Problemy Kibernet, 36 /1979/, 5-22.

[10] S.S. Marcenkov, On homogenous algebras /in Russian/, Dokl. Akad. Nauk SSSR, 256/1981/, 787-790.

[11] S.S. Marcenkov, personal communication.

[12] E.Marczewski, Homogenous algebras and homogenous operations, Fund. Math., 56/1964/, 81-103.

[13] P.P. Pálfy-L.Szabó-Á.Szendrei, Automorphism groups and functional completeness, Algebra Universalis /to appear/

[14] E.L. Post, The two-valued interative systems of mathematical logic, Annals of Math. Studies, 5., Princeton Univ. Press, Princeton, N.J., 1941.

[15] I.G. Rosenberg, Über die funktionale Vollstandigkeit in den mehrwertigen Logiken, Rozpr. Ces. Akad. Ved. Ser. Math. Nat. Sci., 80/1970/, 3-93.

[16] A. Salomaa, On infinitely generated sets of operations in finite algebras, Ann. Univ. Turkuensis, ser. A. 74 /1964/, 1-13.

[17] Á.Szendrei, On closed sets of linear operations over a finite set of square-free cardinality, Elektron, Inform. Kybernet. 14 1978/ 547-559.

[18] Á.Szendrei, Clones of linear operations on finite sets, in Coll. Math. Soc.J. Bolyai 28. Finite algebra and multiple-valued logic /Szeged/, North-Holland 1979, 693-738.

ON FREE SPECTRA OF CLONES WITH SHARPLY TRANSITIVE AUTOMORPHISM GROUPS

by J. Demetrovics and L. Rónyai

Computer and Automation Institute, Hungarian Academy of
Sciences, Budapest, Hungary

Abstract

In this paper we determine the free spectra of clones selfdual to a sharply transitive permutation group. The result is a generalization of the authors' earlier results concerning regular groups, alternating and symmetric groups. The computation is based on extension properties of partial selfdual operations.

Introduction

Let E_k denote the k element set $\{0,1,\ldots k-1\}$, $k \geq 2$. A <u>clone</u> D over E_k is a nonempty set of operations (switching functions) on E_k which contains all projections and is closed under forming arbitrary superpositions. From an algebraic point of view D consists of all polynomial functions of an algebra over the base set E_k, for example those of the algebra $\langle E_k, D \rangle$. The free spectrum of D is the sequence $s_n(D)$, $n \geq 0$, where $s_n(D)$ is the cardinality of the algebra with n free generators in the equational class generated by the algebra $\langle E_k, D \rangle$. From the point of view of multiple valued logic $s_n(D)$ is the number of n-ary functions in the clone D.

The free spectrum $s_n(D)$ is an important invariant of the clone D(see Berman [1] , [2], Grätzer [6]). In [5] we have investigated the free spectra of clones selfdual to various types of permutation groups (semiregular, alternating and symmetric groups). The aim of this paper is to compute the free spectrum of clones consisting of all functions selfdual to a sharply transitive permutation group.

Definitions and notation

All permutation groups are considered to act on the set E_k. Let S_k and A_k be denote the symmetric and alternating groups on E_k, respectively.

<u>Definition</u>: Let $H \leq S_k$ be a permutation group. H is called <u>sharply</u> λ- transitive if for any two sequences $a_1,\ldots,a_\lambda$, $a_i \neq a_j$ and $b_1,\ldots,b_\lambda$, $b_i \neq b_j$, $1 \leq i \neq j \leq \lambda$ there is exactly one $\pi \in H$ for which $\pi(a_i) = b_i$, $1 \leq i \leq \lambda$.

Sharply 1-transitive permutation groups are the regular ones. For $\lambda \geq 2$ there are the following possibilities (Blake-Cohen-Deza [3], Nagao [8]).

$\lambda = 2$ The group of all linear transformations $x \to ax+b$ on a finite near field.

$\lambda = 3$ The group of all transformations $x \to (a.x+b)/(c.x+d)$. Here + and / are the corresponding operations of a finite field and . is either the field or a proper near field multiplication.

$\lambda = 4$ The Mathieu group M_{11} with the usual action ($k=11$).

$\lambda = 5$ The Mathieu group M_{12} with the usual action ($k=12$).

$\lambda = k-2$ The permutation group A_k.

$\lambda = k-1, k$ The permutation group S_k.

<u>Definition</u>: Let $\beta : E_k^n \to E_k$ be a (partial) function and $H \leq S_k$ be a permutation group on E_k. β is called <u>selfdual</u> to the permutation group H if for $\underline{x} = (x_1,\ldots,x_n) \in E_k^n$

$\pi \in H$ $\pi \beta(\underline{x}) = \beta(\pi(x_1),\ldots,\pi(x_n)) = \beta(\pi(\underline{x}))$ holds whenever $\underline{x}$, $\pi(\underline{x}) \in \text{dom } \beta$.

The following lemma concerns with the extension properties of partial selfdual functions.

<u>Lemma 1</u>. ([4], [5]). Let $\beta : E_k^n \to E_k$ be a partial function, selfdual to a permutation group $H \leq S_k$. Then there exists an $f : E_k^n \to E_k$ such that

i. dom $f = E_k^n$,

ii. f is selfdual to H,

iii. if $\underline{x} \in \text{dom } \beta$ then $f(\underline{x}) = \beta(\underline{x})$.

The extension is unique if and only if dom β intersects each H orbit on E_k^n. □

For $H \leq S_k$ let

$D_H = \{f, f$ is an operation on E_k and selfdual to H}. D_H is the clone of all functions selfdual to H, or in other words D_H is the largest clone D over E_k with the property

$$\text{Aut}(\langle E_k, D \rangle) = H.$$

We want to determine the spectrum $s_n(D_H)$, $n \geq 1$ where H is a sharply λ-transitive permutation group on E_k for some λ. From the point of view of multiple valued logic the most important special cases are:

i. k is a prime and $H=\langle \pi \rangle$ where π is a cycle of length k. In this case H is a regular permutation group and D_H is a maximal clone.

ii. $H=S_k$. Then D_H is the clone of all homogeneous functions.

The results

Our main theorem stated as follows.

__Theorem__ Let $k \geq 2$ and $H \leq S_k$ be a sharply λ-transitive permutation group. Then for $n \geq 1$, $\lambda \leq k-1$

$$S_n(D_H) = \prod_{i=1}^{\lambda-1} i^{S(n,i)} \cdot k^{r(n,\lambda)} \quad \text{where}$$

$$r(n,\lambda) = S(n,\lambda) + \sum_{j=\lambda+1}^{k} S(n,j)(k-\lambda)\ldots(k-j+1).$$

Here $S(n,\ell)$ mean Stirling numbers of second kind.

__Corollary 1__ ([5]). If $k \geq 2$ and $H \leq S_k$ is a regular permutation group then for $n \geq 1$

$$s_n(H_D) = k^{k^{n-1}}$$

__Proof.__ From our theorem

$$S_n(D_H) = k^{S(n,1)+r(n,1)} \tag{1}$$

Now let us consider the following identity (Lovász [7])

$$\sum_{\ell=1}^{m} S(m,\ell)x(x-1)\ldots(x-\ell+1) = x^m$$

If we substitute $x=k$ and $m=n$ and divide both sides by k we obtain the following:

$$S(n,1) + \sum_{\ell=2}^{n} S(n,\ell)(k-1)\ldots(k-\ell+1) = k^{n-1}$$

But

$$S(n,\ell)(k-1)\ldots(k-\ell+1) = 0$$

if $\ell > k$ therefore

$$S(n,1) + \sum_{\ell=2}^{k} S(n,\ell)(k-1)\ldots(k-\ell+1) = k^{n-1}$$

and the left side is just the exponent of k in the (1). $\square$

__Corollary 2__ ([5]). Let $k \geq 2$ and $n \geq 1$. Then

$$S_n(D_{S_k}) = \prod_{i=1}^{k-2} i^{S(n,i)} \cdot k^{S(n,k-1)+S(n,k)} \square$$

$$S_n(D_{A_k}) = \prod_{i=1}^{k-3} i^{S(n,i)} \cdot k^{S(n,k-2)+2S(n,k-1)+2S(n,k)} \square$$

In order to prove the theorem we need some preparatory lemmas.

__Lemma 2.__ Let $H \leq S_k$ and $X = \{\underline{x}^1, \ldots, \underline{x}^\ell\}$ be a set of orbit representatives of H on E_k^n. (H acts componentwise on E_k^n.) Let H_i be denote the stabilizer of $\underline{x}^i$ e.g.

$$H_i = \{\pi \in H \; \pi(\underline{x}^i) = \underline{x}^i\} \quad 1 \leq i \leq \ell.$$

Let

$$k_i = |\{y \in E_k, \; \pi(y) = y \text{ for all } \pi \in H_i\}| \quad 1 \leq i \leq \ell.$$

Then
$$S_n(D_H) = \prod_{i=1}^{\ell} k_i$$

__Proof.:__ If $f: E_k^n \to E_k$ then let β_f be the restriction of f to the set X. Thus $\beta_f: E_k^n \to E_k$ is a partial function.

We note that if f is selfdual to H then β_f is selfdual as well and conversely, if β is a function selfdual to H for which dom $\beta = X$ then by lemma 1 there is exactly one $f \in D_H$, dom $f = E_k^n$ with the property $\beta_f = \beta$. Therefore

$$S_n(D_H) = |\{\beta: E_k^n \to E_k, \text{ dom } \beta = X, \beta \text{ is selfdual to } H\}|$$

We note that for every $\underline{x}^i \in X = \text{dom}\beta$, $\pi(\underline{x}^i) \in X$ if and only if $\pi \in H_i$ and thus, β is selfdual to H if and only if for each i, $1 \leq i \leq \ell$ and $\pi \in H_i$, $\pi\beta(\underline{x}^i) = \beta\pi(\underline{x}^i) = \beta(\underline{x}^i)$ holds. There are k_i possible ways to choose the value $\beta(\underline{x}^i)$ and for different values of i we can choose independently.

$$|\{\beta: E_k^n \to E_k, \text{ dom}\beta = X, \beta \text{ is selfdual to } H\}| = \prod_{i=1}^{\ell} k_i$$

and this proves the lemma. $\square$

__Definition__ Let $\underline{x} = (x_1, \ldots, x_n) \in E_k^n$. The __pattern__ of $\underline{x}$ is the partition P of the set $\{1,2,\ldots,n\}$ such that $i \equiv j \pmod{P}$ if and only if $x_i = x_j$. A pattern P is called ℓ-pattern $1 \leq \ell \leq n$, if it has exactly ℓ

127

classes.

The number of different ℓ patterns is $S(n,\ell)$. If $\underline{x} \in E_k^n$ and $\pi \in S_k$ then $\underline{x}$ and $\pi(\underline{x})$ have the same pattern. This implies that for $H \leq S_k$ the elements of an H-orbit on E_k^n have same pattern. So, we can speak about the pattern of an orbit.

__Lemma 3.__ Let $k \geq 2$, $n \geq 1$ and $H \leq S_k$ a sharply transitive permutation group. The number of H orbits on E_k^n having a fixed ℓ-pattern P is 1 if $\ell \leq \lambda$ and $(k-\lambda)(k-\lambda-1)\ldots(k-\ell+1)$ if $\lambda < \ell \leq k$.

__Proof.:__ Let $P_1, \ldots P_\ell$ be the classes of the partition P, and $j_i \in P_i$ $1 \leq i \leq \ell$. If $\ell \leq \lambda$ and $\underline{x}, \underline{y} \in E_k^n$ have the pattern P then
$$x_{j_r} \neq x_{j_s}, \quad y_{j_r} \neq y_{j_s} \quad \text{for } r \neq s.$$

By the λ-transitivity of H there is a $\pi \in H$ for which $\pi(x_{ji}) = y_{ji}$ $1 \leq i \leq \ell$ therefore $\pi(\underline{x}) = \underline{y}$, $\underline{x}$ and $\underline{y}$ belong to the same orbit.

Now let $\lambda < \ell \leq k$ and let $Y = \{\underline{x} \in E_k^n$, the pattern of $\underline{x}$ is P and $x_i = j-1$ if $i \in P_j$ $1 \leq j \leq \lambda\}$. It is clear that $|Y| = (k-\lambda)\ldots(k-\ell+1)$. If $\underline{x} \neq \underline{y} \in Y$ and $\pi \in H$ so that $\pi(\underline{x}) = \underline{y}$ then $\pi(i) = i$ for $0 \leq i \leq \lambda-1$. By the sharp transitivity of H, $\pi = id$, $\underline{x} = \underline{y}$, a contradiction. This implies that $\underline{x}$ and $\underline{y}$ belong to different orbits. By the λ-transitivity of H for every $\underline{x} \in E_k^n$ having the pattern P there is a $\pi \in H$ such that $\pi(\underline{x}) \in Y$. The above observations show that Y represents all orbits of H and the lemma follows. ⊔

__Lemma 4.__ Let $H \leq S_k$ be a sharply λ-transitive permutation group and $\underline{x} \in E_k^n$ have an ℓ-pattern P. Let $\lambda < k$ and
$$K = \{\pi \in H, \quad \pi(\underline{x}) = \underline{x}\}$$
and
$$k(\underline{x}) = |\{y \in E_k, \quad \pi(y) = y \text{ if } \pi \in K\}|$$

Then $k(\underline{x}) = \ell$ if $1 \leq \ell < \lambda$ and $k(\underline{x}) = k$ otherwise.

__Proof.:__ First let $1 \leq \ell < \lambda$. Then K is the stabilizer of $\ell < \lambda$ elements of E_k (the components of $\underline{x}$), hence K is transitive on the remaining $k-\ell \geq 2$ elements of E_k, thus $k(\underline{x}) = \ell$.

If $\ell \geq \lambda$ then by the sharp λ-transitivity of H, $K = 1$, i.e. K fixes all elements of E_k. hence $k(\underline{x}) = k$. □

We are ready to prove the theorem.

__Proof.:__ Let X be a set of orbit represen-

tatives of H on E_k^n. By lemma 2:
$$S_n(D_H) = \prod_{\underline{x} \in X} k(\underline{x})$$

Now let
$$X_i = \{\underline{x} \in X, \ \underline{x} \text{ has an i-pattern}\} \quad 1 \leq i \leq k.$$

By lemma 3 $|X_i| = S(n,i)$ if $i \leq \lambda$ and $|X_i| = S(n,i)(k-\lambda)(k-\lambda-1)\ldots(k-i+1)$ if $\lambda < i \leq k$. Using these facts and the result of lemma 4 we obtain the following:
$$\prod_{\underline{x} \in X} k(\underline{x}) = \prod_{i=1}^{k} \prod_{\underline{x} \in X_i} k(\underline{x}) =$$
$$\prod_{i=1}^{\lambda} k(\underline{x})^{S(n,i)} \cdot \prod_{j=\lambda+1}^{k} k(\underline{x})^{S(n,j)(k-\lambda)\ldots(k-j+1)} =$$
$$= \prod_{i=1}^{\lambda-1} i^{S(n,i)} \cdot k^{S(n,\lambda)} + \sum_{j=\lambda+1}^{k} S(n,j)(k-\lambda)\ldots(k-j+1)$$

The proof is complete. □

References

[1] Berman, J., Algebraic properties of k-valued logics, __Proc. 10. Int. Symp. on Multiple-valued logic__, Evanston Illinois 1980.

[2] Berman, J., Free spectra of 3-element algebras, __Proc. 4. Int. Conf. on Universal Algebra and Lattice Theory__, Puebla Mexico 1982.

[3] Blake, I.F., Cohen, G., Deza, M. Coding with permutations, Information and Control, 43 (1979) 1-19.

[4] Demetrovics, J., Hannák, L., Rónyai, L. Selfdual classes and automorphism groups, this volume.

[5] Demetrovics, J., Rónyai, L., On free spectra of selfdual clones, submitted for publication to the Bulgarian Academy of Sciences.

[6] Grätzer, G., Composition of functions, __Proc. Conf. on Universal Algebra,__ Queens Univ. Kingston Ontario 1969. 1-106.

[7] Lovász, L., __Combinatorial problems and exercises__, Akadémiai Kiadó, Budapest, 1979.

[8] Nagao, H., __Multiply transitive groups__, Math. Dept., California Inst. of Technology, Pasadena California 1967.

(QUASI)TRANSITIVE ALGEBRAS

Lorenzo Peña

Pontificia Universidad Católica del Ecuador
Apartado 2184, Quito - Ecuador

To Professor Jules Varlet

SUMMARY

Quasitransitive algebras are (extensions of) both Kleene and Stone algebras supplied with: two additional, binary, operations: an equivalential operation, and the "overmeet", ^ -which differs from ordinary meet in lacking idempotence, and in the join's not being distributive into it-; and with an additional unary operation, n, which carries every entity into its lower threshold. The filter of dense elements is considered the truth filter, in virtue of the endorsement principle: what is, to some extent or other however small, true is true. The fuzzy sentential calculus *Ap* corresponding to those algebras is shown to be both sound and complete.

§1.- INTRODUCTION

Transitive algebras have been devised by this paper's author in order algebraically to enlighten his system *Aj* of *transitive logic*. Transitive logic is a fuzzy nonarchimedean infinite-valued logic. The four main ideas underlying and prompting that logical approach are:

1.- Properties come in degrees, most of them in infinitely many degrees. Thus, between any two opposite situations there is a transition margin or skirt, wherein *both* opposite states hold but only to some extent. In that transition fringe it both neither is not fails to be the case that the considered state obtains and yet at the same time that state both obtains and doesn't obtain. (Thus, my approach constitutes the juncture of fuzzy-logic and paraconsistent logic -paraconsistent logics being the ones that countenance negation-inconsistent but sound theories-.)

2.- More specifically, with any normal situation, p, there are associated two abutting or threshold situations: one upper threshold, mp, which is its being something very like true that the situation p holds. And a lower threshold, np, which is its being overtrue that the situation p holds. Thus, contrary to other approaches to fuzzy sets, mine sees their logical structure as atomic (rather than strictly dense or Archimedean), though in a generalized sense to be articulated below. (Its non-archimedean nature results from the existence of a minimal truth-threshold which is *infinitely* smaller than any other degree of truth.)

3.- Identity is a fuzzy relation, which explains why identity is such a crisscross of contradictions. No selfidentity is altogether true or

real, there is always some degree of distinction, even between a thing and itself. Those ideas can be traced back to Leibniz and still more to Heracleitus. Furthermore, by regarding selfequivalence as half-true/half-false, we can, in fuzzy logic, secure a lot of alluring results, as the laws of Aristotle, (pDqDN(pDNq), where 'D' means *implication*, /pDq/ being defined as /p.qIp/, where 'I' means equivalence), of Boethius (N(pDNp)) and of counterexample (p.NqDN(pIq)). Selfequivalence or selfidentity is the transition of transitions, a crosspoint.

4.- While, for any two situations, p and q, its being true that p-and-q, p.q, can be seen as the minimal truth-degree among the ones of p and q, instead its being the case that *not only p but also q*, p^q for short, is something normally less true than either p or q, unless of course either p or q is infinitely true -or infinitely false-. Likewise, its being very true that p, Xp -according to my notation-, is, in most cases, less true than p. Zadeh suggested years ago to take, in the real-valued model, $/p/^2$ as a representation of /It's very true that p/. My own approach generalizes that insight, while at the same time avoiding the introduction of an additional function, like √, for '(at least) a little', definig instead /It's (at least) a little true that p/ or /Kp/ for short as /It's not the case that it's very false (=that it's very true that it's not the case) that p/, i.e. as /NXNp/. An interesting parallel emerges between fuzzy logic and modal logic: which reductions of iterated or piled up fuzzy functors are allowed? Is KXp the same as XKp? Is it the same as p *tout court*?

While the answer to those questions is to be searched for from a philosophical perspective, universal algebra can illuminate the issue, by showing which options are viable. Moreover, algebrization of any system of logic yields a great many model-theoretic results and paves the way to further model-theoretic researches.

Accordingly, this paper is devoted to going into some basic algebraic properties of quasitransitive algebras. Quasi-transitive algebras are generalized transitive algebras, differing from the latter in lacking a tensorial operation B (which is meant as standing for 'In all respects' or 'It's really (or truly) true that'). A couple of words will be said on transitive algebras proper at the end of the paper.

There are plenty of problems concerning both transitive algebras and *quasitransitive algebras* -qq.tt. aa., hereinbelow-, which lie beyond the scope of this paper, which only aims at finding out some main fea-

tures of those algebras, while relating them to other, better known, classes of algebras -in particular to Kleene and Stone algebras.

§2.- A HIERARCHY OF ALGEBRAS

This section's purpose is to show how to reach quasi-transitive algebras starting with skew Kleene algebras, which are a class of generalized Kleene algebras. In what follows all binary operations are assciative to the left; no binary operation links more tightly than any other. Intuitively, let 'Π' remind us of 'totally' or 'altogether' ('utterly',etc.); 'L', of '(at least) to some extent' or '(at least) more or less'; 'F', of 'not at all','by no means', 'not in the least'; 'f' of 'somewhat' or 'a bit'; 'S', of 'It neither is nor fails to be the case that'; '+' of 'or'. Furthermore, let a=m0 be the minimal truth or reality threshold, i.e. the infinitesimally real or true, i.e. what is just a jot (or just a whit, or just a trifle) true or real. I take it that its being somewhat true that p means the same as its being more than infinitesimally true that p.

A *skew Kleene algebra* is a structure $\langle A,K\rangle$, where K=$\langle 1,N,+,^\rangle$, 1 being a nullary operation, N a unary operation, and + and $^$ binary operations,with the following properties, for any x,y,z εA (we define x≤y as x+y=y; and x<y as x≤y but x≠y):

01) $x^y+x = x$ 02) NNx = x   03) $x^y = y^x$
04) $x^y^z = x^(y^z)$ 05) x+y = y+x
06) x+y+z = x+(y+z)   07) x+x = x
08) $x+y^z = x^z+(y^z)$ 09) $x^Nx < y+Ny$
10) Nx+Ny ≤ N(x^y)   11) x≤y iff Ny≤Nx   12) $x^1 = x$

If $\langle A,K\rangle$ is a skew Kleene algebra, then: $\langle A,\langle 1,^\rangle\rangle$ is an abelian monoid, whose neutral element is 1; $\langle A,\langle 1,+\rangle\rangle$ is a bounded join semilattice with a greatest element, which is 1.

An *ultra-kleenean algebra* is a skew Kleene algebra, $\langle A,K\rangle$ satisfying the following postulates for every x,y,z εA: 13) N(x+Nx) ≤ y+Ny
14) N(x+N(Ny+Nz)) = N(x+y)+N(x+z)   15) N(Nx+y)+x = x

Clearly, every ultra-Kleenean algebra, $\langle A,\langle 1,N,+,^\rangle\rangle$ is such that $\langle A,\langle 1,N,+\rangle\rangle$ is a Kleene algebra. An ultra-Kleenean algebra, A=$\langle A,K\rangle$, is said to be *stout* iff it satisfies the following condition: there are u, v εA such that: u=Nv; and {xεA:u≤x} is the intersection of all ultrafilters (i.e. maximal proper filters) of A; and {xεA:x≤v} is the intersection of all ultraideals (i.e. maximal proper ideals) of A; and, for any x,y such that u<x<v and u<y<v, the following holds: x^y < N(Nx+Ny).

A *pseudotransitive algebra* is an algebra $\langle A,P\rangle$, P=$\langle 1,N,H,+,^\rangle$, where H is a unary operation, such that, defining Fx as HNx, the four following conditions are fulfilled: 1º) $\langle A,\langle 1,N,+,^\rangle\rangle$ is a stout ultra-Kleenean algebra; 2º) $\langle A,\langle 1,F,+\rangle\rangle$ is a Stone algebra; 3º) the operation N has a fixpoint, ½; 4º) defining 0 as F1, the following postulate holds, for every x,y εA: 16) $F(x^y+F(x.y)) = 0$

Every pseudotransitive algebra is *pseudoatomic* in the following sense: the filter of dense elements, {xεA: Fx=0} has a least element, u; and its dual ideal, {xεA: Hx=0} has a greatest element, v; those elements, u and v, are of course the ones that were considered in the foregoing paragraph. We are going to see that this notion of pseudoatomicity can be generalized in an interesting way.

EQUIVALENTIAL ALGEBRAS

An *equivalential lattice* is an algebra $\langle A,\langle .,+,I\rangle\rangle$ such that: 1º) $\langle A,\langle .,+\rangle\rangle$ is a distributive lattice with zero; 2º) I is a binary operation; henceforth

we shall be using '<' as follows: x<y means that x≤y but xIy=0; then: 3º) there is an element e εA such that for any x,y,z εA, the following postulates are satisfied: 17) 0<e   18) xIy ≤ zIyI(xIz))
19) xIy ≤ x+zI(y+z)   20) xIy ≤ x.zI(y.z)
21) xIyIe+(xIyI0) = e   22) x.zIx+(x.zIz) = e
23) 0<xIy iff x=y

An *equivalential Kleene algebra* is an algebra $\langle A,\langle 1,N,+,I\rangle\rangle$ such that: $\langle A,\langle 1,N,+\rangle\rangle$ is a Kleene algebra -wherein x.y is of course defined as N(Nx+Ny)-; $\langle A,\langle 1,.,+,I\rangle\rangle$ is an equivalential lattice; and the following postulate holds: 24) xINy = NxIy

An *equivalential pseudotransitive algebra* is an algebra $\langle A,\langle 1,N,H,+,^,I\rangle\rangle$ such that: 1º) $\langle A,\langle 1,N,+,I\rangle\rangle$ is an equivalential Kleene algebra; 2º) $\langle A,\langle 1,N,H,+,^\rangle\rangle$ is a pseudotransitive algebra; 3º) e=½, e being the distinguished element such that, for any x, xIx=e, and ½ being the fixpoint of N; 4º) defining 0 as HN1, the following four postulates hold for every x,y εA: 25) $x^xI(y^y) = xIy$
26) xIy ≤ $x^zI(y^z)$ 27) xIy ≤ HxIHy
28) xIy.Fx.y = 0 = Fx.F(xI0)

Next, we define a relation of *pseudocovering*, ≺ , as follows: in an equivalential lattice, x≺y means that x<y and there is at most one element, u, such that x<u and u<y. An equivalential lattice is said to be *strongly pseudoatomic* iff, for any element x thereof, there are two elements, z and y, such that z≤x, x≤y, and z≺y.

A *quasitransitive algebra* is an equivalential pseudotransitive algebra $\langle A,\langle 1,N,H,+,^,I\rangle\rangle$=A meeting the following conditions: 1º) $\langle A,\langle .,.,+,I\rangle\rangle$ -where x.y is defined as N(Nx+Ny)- is a strongly pseudoatomic equivalential lattice; 2º) there is an element aεA such that the following postulates hold (D being the set of dense elements of A); first we define: /nx/ eq /x^Na/; /mx/ eq /NnNx/; /Sx/ eq /x.Nx/; /fx/ eq /F(xIa).x/.   Then, for any elements x,y,zεA:
29) a is the least element of D;
30) $x.yIx+(xImy)+(y.nxIy) = ½$;31) mxInx = xIa+(NxIa)
32) $x^yIa ≤ xIa+(yIa)$;   33) mnxImx+(xI1) = ½;
34) $m(x^x) = mx^mx$, 35) a<½; 36) fSx.fSy ≤ F(x.yI(x^y))

§3.- POSTULATES FOR QUASITRANSITIVE ALGEBRAS

There is a lot of redundancy in the above characterization of quasitransitive algebras. A more elegant set of postulates for a quasitransitive algebra is now put forward. A quasitransitive algebra is an algebra $\langle A,Qt\rangle$, where Qt = $\langle 1,N,H,n,+,^,I\rangle$ where 1 is a nullary operation, N, H and n are unary operations, and +,^,I are binary operations, satisfying the 24 postulates below. First, let's introduce some definitions: /0/ eq /N1/   /Sx/ eq /x.Nx/   /x.y/ eq /N(Ny+Nx)/ /½/ eq /1I1/   /Fx/ eq /HNx/   /Xx/ eq /x^x/   /a/eq/m0/ /xDy/ eq /x.yIx/   /fx/ eq /F(xIa).x/   /Lx/ eq /NFx/ /mx/ eq /NnNx/   We also introduce two ordering relations: x≤y means that y = y+x; x<y means that, while x≤y, xIy=0. Let D be {xεA: Fx=0}

Postulates (for any x,y,z,u,v εA)
(01) y.x+x = x   (02) xIy ≤ x.u+zI(y+z..u+z)
(03) Hx.Hy = LH(y.x)   (04) zIy ≤ Hx+HzIH(x+y)
(05) vIy ≤ $v^(x.u)^zI(u^z.(x^z)^y)$ (06) $x^1 = x$
(07) $x^y ≤ y.x$ (08) $x.y.F(x^y)= 0$ (09) xIyεD iff x=y
(10) ½ = N½   (11) xIy ≤ zIyI(xIz)   (12) xIy.Fx.y = 0
(13) $F(xI0+x) = 0$ (14) xIyI½+(xIyI0) = ½
(15) XxIXy = xIy   (16) xDy+(yDnx)+(xImy) = ½
(17) F(nmxInx).x = 0   (18) $x^yIa ≤ xIa+(yIa)$
(19) mXx = Xmx   (20) nx = x^n1
(21) nxImx = xIa+(xINa)   (22) a<½
(23) fSx.fSy ≤ F(x.yI(x^y))   (24) NxIy = xINy

130

The remaining of this Section will be devoted to proving a number of properties of qq.tt.aa. as follow from the 24 postulates above. To start with, let's prove some elementary results.

Theorem 01.- In every q.t.a. (quasitransitive algebra) the following hold:

T01 xIx ϵ D (Proof: (09))
T02 1.x = x (Proof: (07),(06),(01))
T03 NNx = x (Proof: T01,(24),(09))
T04 N(y.x)=Nx+Ny   N(y+x)=Nx.Ny   N(Ny.Nx)=x+y
(Proof: T03)   T05 x+0 = x (Proof: T03,T04)
T06 x+x = x = x.x (Proof: T02,(01),T03,T04)
T07 x.(x+y) = x (Proof: (01),T03,T04)
T08 LHx = Hx (Proof: (03),T06)
T09 FFx = NFx (Proof: T08,T03)
T10 Hx.Hy = H(y.x) (Proof: (03),T08)
T11 Fx.Fy = F(x+y) (Proof:T10,T04)
T12 If x$\leq$y and Fx=0, then Fy=0 (Proof: T11,T05,(01))
T13 Hx+Hy =H(x+y) (Proof: T12,(04),T01,(09))
T14 Fx+Fy = F(y.x) (Proof: T13,T04)
T15 F(Fy.Fx) = N(Fy.Fx) (Proof: T14,T09,T04)
T16 1$^\smallfrown$x = x (Proof: (05),T01,T12,(06),T06,(09))
T17 x.y = y.x (Proof: (05),T01,T12,T16,(06),(09))
T18 x+y = y+x (Proof: T03,T04,T17)
T19 HHx = Hx (Proof: T03,T08)
T20 Lx = HLx (Proof: T09)
T21 Fx = FLx (Proof: T03,T19)
T22 xϵD iff LxϵD (Proof: T21)
T23 Lx = FFx (Proof: T09)

Then follow a number of corollaries that I set out without proof: T24 L(x+y) = Lx+Ly

T25 L(x.y) = Lx.Ly   T26 x.y+z = x+z.(y+z)
T27 x+y.z = x.z+(y.z)   T28 x+y+z = x+(y+z)
T29 x.y.z = x.(y.z)   T30 xIy $\leq$ x+zI(y+z)
T31 xIy $\leq$ x$^\smallfrown$zI(y$^\smallfrown$z)   T32 x$^\smallfrown$y = y$^\smallfrown$x
T33 x$^\smallfrown$y$^\smallfrown$z = y$^\smallfrown$z$^\smallfrown$x   T34 x$^\smallfrown$y$^\smallfrown$z = x$^\smallfrown$(y$^\smallfrown$z)
T35 x.Fx =0   T36 x $\leq$ x+y   T37 x.y $\leq$ x
T38 xI1 $\leq$ x.zIz   T39 xI1 $\leq$ x+zIx   T40 xI0 $\leq$ x.zIx
T41 xI0 $\leq$ x+zIz   T42 x.y=0 iff: either x=0 or y=0
T43 x.y=1 iff: both x=1 and y=1   T44 x+y=0 iff:
both x=0 and y=0   T45 x.y=0 iff: either x=0 or y=0

Theorem 02.- In any q.t.a., let p and q be two polynomials; then F(p+q) = 0 iff F(Lp+q) = 0 (Proof: T11,T21). *Corollary:* For any x, xϵD iff Lx ϵ D (Proof: T06 of Thm01)

Theorem 03.- In a q.t.a., if x$\leq$y and y$\leq$z, then x$\leq$z. (Proof: in virtue of equations proved in Thm01).

Theorem 04.- In any q.t.a. F0=1 and F1=0. (Proof: (13),T09). *1st Corollary:* H1=1 , H0=0. *2d Corollary:* FF0=0 , FF1=1. *3d Corollary:* F(x+Fx)=0 (Proof: T35, T14,T17, Thm02,Thm04). *4th Corollary:* F(Fx+(Fy+(x.y)) =0. (Proof: T14,T17 and T28 of Thm01).

Theorem 05.- In any q.t.a., if x+y ϵ D, Fx+u ϵ D, Fy+v ϵ D, then u+v ϵ D (Proof: let's suppose F(x+y)= = 0 = F(Fx+u) = F(Fy+v); Then: Lx+Ly = 1 = Fx+Lu = = Fy+Lv; Fx.Fy+(Lu+Lv) = Fx+Lu+Lv.(Fy+Lv+Lu); then: Fx.Fy+(Lu+Lv) = 1 ; Hence: Lu+Lv = 1. Therefore, in virtue of Thm02, and Thm04, u+v ϵ D. *1st Corollary:* In any q.t.a., if x+y ϵ D and Fy+z ϵ D, then x+z ϵ D. (Proof: 3d Corol of Thm04). *2d Corollary:* In any q.t.a., if Fx=0=F(Fx+y), then Fy=0.

Theorem 06.- In any q.t.a. the following holds: if Fx+y ϵ D, Fy+z ϵ D, then Fx+z ϵ D. (Proof: Thm05 and 3dCorol of Thm04).

Theorem 07.- In any q.t.a., the following holds: xIy = yIx (Proof: (11),T01 and T12 of Thm01,(09)).

Theorem 08.- In any q.t.a., let p and q be two polynomials. Then, Fp+q ϵ D iff pI0+q ϵ D. (Proof: in virtue of (13), Them02 and T23 of Thm01, FFx+(xI0)ϵD.

Thus, in virtue of the 1stCorol of Thm05, Fp+q ϵ D only if pI0+q ϵ D. On the other hand, F(xI0)+Fx ϵ D, in virtue of (12), Thm07, 2dCorol of Thm04,T05 -as well as other equations- of Thm01. Therefore, in virtue of 1stCorol of Thm05, pI0+q ϵD only if Fp+q $\epsilon \Gamma$.)

Theorem 09.- In any q.t.a., $\frac{1}{2}\epsilon D$. (Proof:T01 of Thm01)

Theorem 10.- In any q.t.a., xIx=$\frac{1}{2}$. (Proof: T01 and T05 of Thm01, Thm08,Thm09,(14) and (09)). *Corollary:* x=y iff xIy = $\frac{1}{2}$ iff xIyI$\frac{1}{2}$ ϵ D iff xIy ϵ D.

Theorem 11.- In any q.t.a., $\leq$ is an antisymmetric relation. (Proof: Let's suppose that x$\leq$y and y$\leq$x. Then, in virtue of the equations proved in Thm01, x+y = y = x.y = x.) *Corollary:* In any q.t.a.,1 is the greatest element, while 0 is the smallest element.

Theorem 12.- In any q.t.a., if x$\leq$y, then Fx+y ϵ D. (Proof: x$\leq$y iff x+y=y. Hence, if x$\leq$y, then Fx+y = Fx+(x+y). But Fx+(x+y) = Fx+x+y ϵ D -in virtue of 3dCorol of Thm04, T36 of Thm01, and T12 of Thm01.) *Corollary:* Whenever x$\leq$y and Fy+z ϵ D, then Fx+z ϵ D. And whenever Fx+y ϵ D, y$\leq$z, then Fx+z ϵ D. (Proof: Thm12 and Thm06).

Theorem 13.- In any q.t.a., Lx+(xDy) ϵ D for any x and y ϵ A. (Proof: Thm12, df of D,Thm02,(13), and T40 of Thm01.) *Corollary:* Fx+(FxDy) ϵ D. (Proof: T21 and T23 of Thm01).

Theorem 14.- Let /xCy/ be defined as /Fx+y/. Then, in any q.t.a., the following hold: (C1) xC(yCx) ϵ D (C2) xC(yCz)C(xCyC(xCz)) ϵ D (C3) xC0C0Cx ϵ D (Proof: for (C1): 3dCorol of Thm04,T18,T28,T12 and T36 of Thm01. For (C2): 3dCorol of Thm04, T18,T28 and T36, T12 and T26 of Thm01, and 4thCorol of Thm 04. For (C3): 3dCorol of Thm04 and T18,T05,T21 and T23 of Thm01). *1st Corollary:* Let <A,Qt> = A be a q.t.a., and let θ be a <0,.,.,+,F>-congruence on A. Then A/θ is a Boolean algebra. (Proof: in A, for any x and y (taking pθq as meaning: p is congruent with q modulo θ), (x+y)θ(xC0Cy) and (x.y)θ(xC0+ (yC0)C0). Of course, the unit element of A/θ is the cokernel of θ.) *2d Corollary:* Let Lc be a system <V,F>, where V is a set of symbols and F is a set of formulas generated from those symbols in accordance with some rules of formation; and let V contain only, in addition to sentential variables, these four symbols: '0', 'F','+','.', so that, if "p" and "q" ϵ F, then '0', "p+q", "p.q" and "Fp" ϵ F. Let: A be a q.t.a. Let's define a valuation from Lc into A as a mapping, v, carrying formulas of Lc into elements of A and such that, for every "p" and "q" ϵ F, $v(0)$=0, v(Fp) = $F(v(p))$, v(p+q) = v(p)+v(q), and v(p.q) = v(p).v(q) (no equivocality is warranted concerning our twofold use of '0','F','+' and '.' both as symbols of Lc and as operations of A). Then a formula, "p", of Lc is said to be valid in A iff every valuation v Lc$\rightarrow$A is such that v(p) ϵ D. A formula of Lc is said to be valid iff it is valid in every q.t.a.

Then a formula of Lc is valid -in accordance with the foregoing characterization- iff it is a theorem of (a system of) classical logic, i.e. of truth-functional two-valued logic. Proof: (If) is an immediate corollary of Thm14.

(Only if): Suppose that some formula of Lc, "p", is valid -in accordance with our definition above- without being a theorem of classical logic; the validity of "p" depends not on what values are assigned to the sentential variables lying in "p", but only on the regulations laid down as regards the symbols '+', '.','F' and '0'; thus every q.t.a. A is such that for every valuation v from Lc into A, v(p) ϵ D in virtue of some property or other of at least one of

131

those four operations. So, we can form a sentential calculus on Lc having as its axioms the ones of any classical system plus "p", and as its rules of inference modus ponens and substitution. But -as is well-known, from the completeness of classical sentential logic- such a calculus would be *deliquescent*, i.e. such that for every formula "q" of Lc, "q" would be a theorem of the envisaged calculus. But that would entail that in any q.t.a. every $x \varepsilon A$ was such that $x \varepsilon D$, which is downright false, since there are q.t. a. with more than one element, and in any such q.t. a. $0 \notin D$.

Theorem 15.- In any q.t.a. xDyC(xC(xDy)) ε D and xC(xDy)C(xDy) ε D. The proof as regards the former polynomial is straightforward in virtue of 3dCorol of Thm04 and T18,T28 and T12 of Thm01. As regards the latter polynomial, the proof uses Thm14, Thm13 (plus T23 of Thm01), as well as the 2dCorol of Thm05.

Theorem 16.- In any q.t.a. x.y ε D iff both $x \varepsilon D$ and $y \varepsilon D$. (Proof: either via Thm14 or via T14, T17 and T44 of Thm01).

Theorem 17.- In any q.t.a., xC(y.z) = xCy.(xCz); xC(y+z) = xCy+(xCz); x+yCz = xCz.(yCz); x.yCz = xCz+(yCz). (Proof: the equalities proved in Thm01).

Theorem 18.- In any q.t.a., xIy.(xIz)C(yIz) ε D. (Proof: Thms 16&17,(12), the equations of Thm01, the Corol of Thm02, as well as (11), Thm12,Thm06,Thm07).

Theorem 19.- In any q.t.a., both HxC(HxIx) and FxC(FxINx) ε D. (Proof: Thms18&01,(13),Thms02,04,16, 17&18.) *Corollary:* HxDx, FxDNx, xDLx εD. (Pr:Thm01&15)

Theorem 20.- In any q.t.a. the following hold: xDy= =NyDNx; xDy+(yDx) εD; xDyC(xCy) εD; xD(y+z) = xDy+(xDz); xD(y.z) = xDy.(xDz); x.yDz = xDy+(xDz); x+yDz = xDy.(xDz); x.yDx εD; xDy.(yDx) = xIy; if p and q are polynomials formed with + and/or . only, out of equivalential or implicational polynomials -i.e. of polynomials of the form rIs or rDs, respectively- then, both pCqC(pDq) and pCq.(qCp)C(pIq) εD. For brevity sake, I omit the proof of this theorem, as well as the ones of other theorems below.

Theorem 21.- In any q.t.a. there is just one fixpoint for N, viz. ½. Proof: Suppose there are two such fixpoints, ½ and, say, e. Then, e=Ne; thus, in virtue of Thm20: eD½+(½De) εD. Hence, eD½+(NeDN½) εD, and consequently eD½ εD. We likewise prove ½De εD. Whence it follos that ½Ie εD, and accordingly ½=e.

Theorem 22.- In any q.t.a. all the following elements are dense, i.e. members of D: xDNxDNx, NxDxDx, xIxI(yIy), xDyDN(x.Ny), xDN(xDNx), N(xDNx), xDyDN(xDNy), xDyDN(yDNx), xDyDN(NxDy), xDyDN(yDx), xDyD(xDNyDNx), xDyD(xDN(xDNy)), xDN(xDSy), xDSyDNx, SxDNSy. Furthermore, if p is a polynomial wherein there is an occurrence of x, let q be the result of replacing that occurrence of x in p by an occurrence of y; them p.NqDN(xIy) ε D. This I call the *principle of distinction*.

Theorem 23.- The system of postulates set out hereinabove is equivalent to several systems of postulates, wherein, instead of defining ½ as 1I1, we take ½ as a primitive, and we replace every occurrence of ½ in postulates (14)&(16) by an occurrence of 1I1, and then we add, from the postulates below: either both (h01) and (h02); or else both (h01) and (h03); or else both (h01) and the principle of distinction; or else both (h01) and (h04), even deleting (10); or else both (h01) and (h05), even deleting (10); or else both (h01)&(h06); or else both (h01)&(h07).

(h01) N(xIx)D(xIx) ε D (h02) xDNxDNx ε D
(h03) xDyDN(x.Ny) ε D (h04) xDyDN(xDNy) ε D

(h05) xDyDN(yDx) ε D (h06) xDyD(xDN(xDNy) ε D
(h07) xDSyDNx ε D (Incidental Comment.- Let's call *Heracleitian algebra* any Stone algebra with an equivalential operation satisfying such postulates as (02),(04) -if the algebra contains a topological operation like H-, (09),(11),(12),(13),(14) and xDy+(yDx) εD, provided it also satisfies the postulate xIx = N(xIx). Every q.t.a. is an Heracleitian algebra. Philosophically, that means that *within* a q.t.a. selfequivalence is as true as untrue, which, as I have it, is Heracleitus' principle.)

§4.- THE FILTER OF DENSE ELEMENTS D

The Glivenko filter of dense elements enjoys a privileged role in any Stone algebra. Since any q. t.a. is an extension of a Stone algebra, we need to go into the Glivenko filter's properties in qq.tt.aa

Moreover, as against *alethic maximalism* which withholds the label of *true* from whatever is not completely or absolutely true -a stand taken both by classical logic upholders and by such fuzzy-set theorists as feel bound to junk the principle of excluded middle-, my approach relies on the *principle of endorsement*, to be: whatever is more or less true is true. Thus, according to my lights the filter D of dense elements is *the* truth-filter. Whatever corresponds to a dense element is more or less true; hence, it's true (*tout court*). Being true is by no means the same as being downright or wholly true. However, the truth filter is negation-inconsistent. There are mutually contradictory truths.

Theorem 24.- In any q.t.a., $D = \{x \varepsilon A: xI0=0\}$.(Proof: if xI0=0, then Fx=0, in virtue of (13). On the other hand, in virtue of (12) F(xI0)+Fx εD (see proof of Thm08); thus, if Fx=0, F(xI0) εD, hence L(xI0) = 0, hence xI0=0, since, for any z, z $\leq$ Lz -cf. Corol of Thm19). *Corollary:* $D = \{x \varepsilon A: 0<x\}$

Theorem 25.- In any q.t.a. a is the smallest dense element, i.e. the least element of D. Let's split the proof: First, we prove that a is dense, as follows: FaC(aI0) εD, in virtue of (13), Thm02 and T23 of Thm01. Now, aI0C(nxImx) εD, for every x, in virtue of (02),(05),(20) and Thm12. In virtue of (21) that entails that aI0C(xIa+(xINa)) εD which can be easily proved to be equivalent to aI0C(xI0+(NxI0)) εD whence it follows that aI0C(½I0) εD and aI0C(½I1) εD which, in virtue of Thm16&17, entails that aI0C(½I0..½I1). Now, in virtue of Thm18, ½I0.(½I1)C(0I1) εD. Applying Thm06 to the results just reached, we get at FaC(0I1) εD. Since 1 is dense, that means -in virtue of Thm24- that FaC0 εD, and thus FFa εD; therefore $a \varepsilon D$.

Now, we sketch the proof of every dense element, x, being such that a≤x, which in virtue of ≤ being an ordering relation, entails that a is a lower bound of D. In virtue of (16),(20),(06) and T32 of Thm01, we have that 1Dx+(xDn1+(1IN(Nx^n1))) εD, whence it follows that NxC(xDn1) εD, so xC(aDx) εD, since Nn1 = a. Suppose $x \varepsilon D$; then, in virtue of 2d Corol of Thm05, aDx εD, which means that a≤x, in virtue of (09). *1st Corollary:* $D = \{x \varepsilon A: aDx=½\} = $ [a) $= \{x \varepsilon A: F(aDx)=0\}$. Accordingly, in any q.t.a. D is the *principal filter* generated by m0. *2d Corollary:* 0<a. (Proof: Corol of Thm24).

Theorem 26.- In any q.t.a. with more than one element, < is a strict ordering relation, i.e. transitive and asymmetric. (Proof: Since any q.t.a. has a 1, if it has at least two elements, then 1≠0. Moreover, ½≠0 and ½≠1, since ½=N½. Suppose now that x<y and y<z. Then we have: x+y = y, z = z+y; thus,since

xIz ≥ x+yI(z+y) (T30 of Thm01), we conclude that
xIz ≥ yIz; whence it follows xIz=0; therefore, x<z.
Thus, we've proved that < is transitive. That < is
asymmetric is shown as follows: suppose x<y and y<x;
then x≤y and y≤x, and xIy=0; so, x=y, but xIy=0, i.
e. ½=0, which blatantly runs counter to our previous
result, viz. ½≠0 (in any q.t.a. with more than one
element)

Theorem 27.- In any q.t.a. with more tha n one ele-
ment, its *trunk*, i.e. the convex set {xεA: a<x<Na}
-which we'll abbreviate as T -satisfies neither the
descending chain condition nor the ascending chain
condition. Proof: as shown on top of the proof of
Thm26, 0≠1; thus 0≠½≠1, since ½=N½. But ½εT in vir-
tue of (22). Since ½=N½, we have that S½=½, and
S½Ia=0; thus, F(S½Ia)=1, and consequently fS½ = ½εT;
but of course T is a subset of D. In virtue of (23)
and T12 of Thm01, F(½IX½) εD, so ½IX½=0. Since, for
any y and z, y⁻z≤y (in virtue of (07)),Xx≤x, so X½≤x;
therefore X½<½. Now, for every x and y, if x,y εT,
x⁻y εT, in virtue of (08),(16),(07). Accordingly,
X½ εT, whence it follows that XX½ < X½ and XX½ εT,
and so on (the result is generalized by mathematical
induction). Thus, we get at a descending chain: ½,
X½,XX ,XXX½,... The chain has a g.l.b., to wit: a=m0.
But of course a doesn't belong on the chain; thus,
the chain has no minimal element. Therefore T is not
Noetherian and the algebra under consideration is
not Noetherian either. Furthermore, let's definitio-
nally introduce /Kx/ as follows: /Kx/ eq /NXNx/. Then,
it can in the same way be shown that T doesn't satis-
fy the ascending chain condition, since T contains
this ascending chain: ½<K½<KK½<KKK½<... The l.u.b.
of that chain is Na=n1, which is not a member of T.
Corollary: Any q.t.a. wherein there are at least two
elements comprises infinitely many elements, such
that for any two of themm x and y, x<y.

Theorem 28.- Every q.t.a. is strongly pseudoatomic.
Proof: Let's definitionally introduce an operator %
as follows: /x%y/ eq /xDy.F(xIy)/. It can very easi-
ly be proved that x%y εD iff x<y, and that x<y.z iff
both x<y and x<z; and that y+z<x iff both y<x and z<x.
Let's now definitionally introduce two new opera-
tions as follows: /ñx/ eq /L(NxIa+fSx+H(x+Nx)).nx/
/m̃x/ eq /L(xIa+fSx+H(x+Nx)).mx+L(xINa)/
Then, it can be proved that, for any x, ñx≤x, and
x≤m̃x, and ñx%m̃x.(ñx%(y.z).(y+z%m̃x)C(yIz) ε D. There-
fore, ñx<m̃x; and, if ñx<y.z -if,that is, ñx<y and
ñx<z- and y+z<m̃x -if, that is, y<m̃x and z<m̃x-, then
y=z. Accordingly, every q.t.a. is strongly pseudoato-
mic.(Notice that we have not proved that, for every
x, ñx<x, nor that x<m̃x. That is not the case: ñNa =
Na, while m̃a=a. And there are many other elements x
such that either x=ñx or x=m̃x; whenever fSx εD, then
nx=ñnx and mx=m̃mx).

§5.- MORPHISMS AND CONGRUENCES

Theorem 29.- Let h be a morphism from a q.t.a., A,
into another q.t.a., A′. Let ½ be the fixpoint of N
in A, and ½′ the fixpoint of N in A′. Then h½=½′.
(Proof: Nh½=hN½=h½; thus, h½ is a fixpoint of N in A.
Since the fixpoint of N in any q.t.a. is unique -in
virtue of Thm21-, h½=½′.)

Theorem 30.- Let θ be a congruence on a q.t.a., A.
Then: (xIy)θ½ iff xθy. (Proof: (Only if): Since θ is
a congruence, let's define an epimorphism, φ, of A
onto A/θ such that φ(x)= ⌈x⌉θ. If (xIy)θ½, φ(xIy) =
φ½ = φxIφy; thus, since, in virtue of Thm29, φ½ is
the only fixpoint of N in A/θ, φx=φy, which means
that ⌈x⌉θ = ⌈y⌉θ, and therefore xθy. (If): If xθy,

then (xIy)θ(yIy); now, yIy = ½.) *1st Corollary:* Let θ
be a congruence on a q.t.a., A. Then, xθNx iff xθ½.
(Proof: (If) is obvious. (Only if) follows from the
theorem plus the fact that in any q.t.a. xINx = xI½).
2d Corollary: Let h be a morphism from a q.t.a., A,
into another q.t.a., A′. Then, h(x)=h(y) iff h(xIy)=
½′, ½′ being the only fixpoint of N in A′.

Theorem 31.- No congruence θ≠ι on a q.t.a., A, with
more than one element is such that xθy while x<y.
(Proof: If xθy, then, in virtue of Thm30, xIyθ½; now,
if x<y, then xIy=0, then ½θ0, which entails that θ=ι,
against the hypothesis). *1st Corollary:* Every congru-
ence θ≠ι on a q.t.a., A, comprising more than one el-
ement has infinitely many congruence classes. (Proof:
Corol of Thm27). *2d Corollary:*Let h be a morphism
from a q.t.a. A = <A,Qt> into another q.t.a. A′. Let
x,y εA and x<y. Then, if hx=hy, A′ comprises just
one element, 0′ -and consequently h is an epimorphism
such that, for any xεA, hx=0′.

DEFINITIONS.- A q.t.a. will be said to be *scalar* iff
it's totally ordered by ≤. A q.t.a. will be said to
be equivalentially scalar iff for any x,y,either
xIy=0 or else xIy=½.

Theorem 32.- For any q.t.a., A, the following are
equivalent: 1) A is scalar; 2) A is equivalentially
scalar; 3) For any two members of A, x and y, either
x=y or x<y or y<x; 4) There is no x≠0 such that Fx≠0.
(Proof: 1)→2). Let x≤y. If 0≠xIy≠½, neither a≤xIy
(since, for any u and v, uIv εD iff uIv=½) nor xIy≤a
(since, for any u and v, u≤v iff uDvεD,and xIyDaCF(xIy)
ε D; thus, were xIy≤a, F(xIy) εD, and so xIy=0.
2)→3). Let's suppose that xIy=0 but that neither x<y
nor y<x. Then neither x≤y nor y≤x; hence x+yIy≠½ and
x+yIx≠½; therefore -since A is equivalentially sca-
lar- x+yIy=0=x+yIx, whence it follows that xDy+(yDx)
=0. Now, in virtue of Thm20, that result is impos-
sible. 3)→4). Suppose there is an x≠0 such that
Fx≠0. Then x≠a, hence either x<a or a<x; the latter
is impossible, since otherwise x would be dense (Thm
25). But x<a entails x=0. Thus, the hypothesis was
impossible. 4)→1). If there is no x≠0 such that
Fx≠0, then for no x and y is it the case that, while
xDy≠½, xDy≠0. Then either xDy=½, which means that
x≤y, or else xDy=0, which entails (see Thm20, where-
in it's said that xDy+(yDx) εD) that yDx εD, which
in turn entails y≤x.)

Theorem 33.- Let A be a scalar q.t.a. and let A′be a
nonscalar q.t.a. Then there is no epimorphism from A
onto A′. (Proof: Suppose there is an epimorphism
A→A′,h. Now, in any q.t.a. x<y iff x%y=½. Suppose
that there are two noncomparable elements of A′, x′
and y′, and that there are two elements of A, x and
y, such that hx=x′ and hy=y′. Now, in virtue of Thm
32, either x%y=½ or y%x=½. Suppose the former. Then
h(x%y)=h½. But, according to Thm29, h½=½′, i.e. the
fixpoint of N in A′. Thus hx%hy=½′, i.e. x′%y′=½′,
which means that x′<y′, which runs counter to the
hypothesis.)

Theorem 34.- Any morphism from a scalar q.t.a. A into
another q.t.a. A′, is a monomorphism, unless A′ com-
prises just one element. (Proof: Thms 31&32.)

Theorem 35.- The free algebra with r generators (for
any 2≤r) associated with the class of qq.tt.aa. is a
nonscalar q.t.a. (Proof: Since all postulates (01)
thru (24) are either identities or conditional iden-
tities -which is the case as regards (09)-, the class
Γ of qq.tt.aa. is closed with respect to formation of
subalgebras and direct products. Therefore, according
to Birkhoff's proof, F_r(Γ) (i.e. the free algebra

with r generators associated with Γ) εΓ, which entails that $F_r(Γ)$ is a free q.t.a. (To be sure, $F_r(Γ)$ is a subdirect product of subalgebras of all qq.tt. aa.) That $F_r(Γ)$ is nonscalar is proved like this: by construction, $F_r(Γ) = W_r(Qt)/θ(Γ)$, where: $W_r(Qt)$ is the free word algebra with r generators on Qt, i.e. on the set of operations of qq.tt.aa., viz $<1,N,H,n,+,\hat{},I>$; and $θ(Γ)$ is a congruence on $W_r(Qt)$ constructed as follows: let Δ be the set of functions δ from the set of r generators of $W_r(Qt)$ into (the set of members of) any q.t.a., defining, for any polynomial $p(x^1,...x^r)$, /δp/ eq /p(δx^1...δx^r)/, then we let /pθq/ mean that δp=δq for every δεΔ. By construction, every such δ is a morphism. θ is the intersection of the kernels of all those morphisms; thus, the only identities holding in $F_r(Γ)$ are the ones that are bound to obtain in virtue of the 24 postulates. Now, for any y,x εT (for any two elements, y,x, of the trunk) of any q.t.a. A, we do know, in virtue of (23), that x^y<x.y, and that x^y < K(x^y); but whether or not x.y ≤ K(x^y) is not settled by the 24 postulates. Nor do those postulates determine whether or not, for any x,y εT, X(x.y)≤x^y, whether or not XKx≤x; in fact, those postulates entail no reduction of iterated or piled up opertaions K, and X, or of any of them confronting ^ or ‡, where /x‡y/ eq /N(Nx^Ny)/. *Corollary:* There is no epimorphism from any scalar q.t.a. onto $F_r(Γ)$, where Γ is the class of qq.tt.aa. (Proof: Thm33)

§6.- SOME EXAMPLES OF QQ.TT.AA.

First, I'm going to construct a scalar q.t.a. as follows. Let's take the set of standard integers, Z. For any such integer, x, nx is: x-1, if x≡0(mod3); x-2,if x≡1(mod3); x, if x≡2(mod3). For any two standard integers, x and y such that x≤y, x^y will be x-3.

We now add to the standard integers two nonstandard natural numbers whatever, say: ∞ and ∞⊕1 -where ⊕ is addition-, as well as their respective negative counterparts, -∞ and -(∞⊕1); the union of Z with the set of those four nonstandard integers will be called *the set of pseudointegers*. For each pseudointeger, x, Nx is defined as -x; moreover, for any two pseudointegers, x and y, x+y is defined as *max*(x,y). If x=∞=y, then x^y=x. If x=∞ and y is a standard integer congruent with either 0 or 1, modulo 3, then x^y=y^x is y-1, if y≡0(mod3), but y-2 if y≡1(mod3). If x=∞ and y is a standard integer congruent with 2 modulo 3, then x^y = y^x = y.

Let x be -∞ and x≤y; then x^y=y^x=x. Finally, if x=∞⊕1, then x^y=y^x=y while, if x=-(∞⊕1), then x^y=y^x=x. If x=∞⊕1, then nx=∞; if x is either ∞ or -∞ or -(∞⊕1), nx=x.

The operation H is defined as follows: if x=∞⊕1, then Hx=x; otherwise, Hx=-(∞⊕1). The operation I is defined like this: if x=y, then xIy=0; elsewise, xIy= -(∞⊕1). The algebraic unary operation 1 has as its value, for any argument, ∞⊕1.

The set of pseudointegers, Z̃ with those operations defined thereon is a scalar q.t.a. For any n≥2, the direct product Z̃^n is a nonscalar q.t.a.

Another q.t.a. is the set of *alethic numbers*, i.e. the interval of hyperreals $[0,1]$ constructed as follows: let's take all real numbers and add any nonstandard positive integer, say ∞. Then we take as the set of alethic numbers such x, 0≤x≤1, as are either a standard real, or else the result of either adding 1/∞ to, or subtracting 1/∞ from a standard

real. The operations on that set of alethic numbers are defined as follows: x+y = *max*(y,y); Nx=1-x; 1(the algebraic nullary operation) = 1(the real number); Hx=1 iff x=1, otherwise Hx=0; if x is either a standard ≠0, y, or y⊕(1/∞), where y is standard, provided y≠0, then nx = y-(1/∞); otherwise, nx=x; if x and y are standard, then x^y=x×y; if one among x,y is either a standard u≠0, or u⊕(1/∞), u being standard, while the other = z⊕(1/∞), z being standard, then x^y = (u×z)⊕(1/∞); if either x or y is 0, then x^y=y^x=0; if x=Nn1, and y≠0, then x^y=y^x=x; the remaining case is the one wherein one among x,y is u-(1/∞), u being a standard while the other among x,y is either v, or v⊕(1/∞), or v-(1/∞), v≠0 being standard; then x^y = (u×v)-(1/∞). Finally, xIy=½ iff x=y; otherwise, xIy=0. Of course, any direct product of Ã (Ã being the algebra of alethic numbers) is a q.t.a. Ã is a scalar q.t.a. And, to be sure, the direct product Πtε T(A_t), where every tεT is either Ã or Z̃, or any other q.t.a. for that matter, is a nonscalar q.t.a., too.

§7.- THE SYSTEM OF TRANSITIVE LOGIC *Ap*

Another q.t.a. is the Tarski algebra of classes of formulas of the sentential calculus *Ap*, which is defined as follows. *Ap* is a structure $<V,F,T,R>$ where: V is its vocabulary; F, the set of its formulas; T, a proper subset of F, viz the class of theorems; and R a set of one inference rule. $V = \{a,H,‡,\hat{},I,(,)\}$. F is determined by these rules: 1.- aεF 2.- If "p"εF and "q"εF, then"p‡q","p^q","pIq","Hp" ε F. The letters 'p','q', etc. are used as schematic letters. In restoring omitted parentheses, a dot written immediately following an occurrence of a two-place functor stands for a left parenthesis whose right mate is to be placed as far to the right as possible. Remaining ambiguities are dispelled by associating leftwards. We'll avail ourselves of the 17 following abbreviations: /Np/ eq /p‡p/ /½/ eq /aIa/ /Fp/ eq /Hnp/ /0/ eq /N(aI½‡F(½IN½))/ /1/ eq /N0/ /p+q/ eq /N(p+q)^1/ /pCq/ eq /Fp‡q/ /p.q/ eq /N(Np+Nq)/ /pDq/ eq /q.pIp/ /Sp/ eq /p.Np/ /np/ eq /p^Na/ /mp/ eq /NnNp/ /Xp/ eq /p^p/ /Lp/ eq /NFp/ /Yp/ eq /pIa.p/ /fp/ eq /FYp.p/ /p‡q/ eq /pDq.F(qDp)/ READINGS: a: The infinitesimally true or real exists p‡q: Neither p nor q. p^q: Not only p but also q. pIq: It's as true that p as that q ≅ The fact that p is equivalent to the fact that q. Np: It's not the case that p ≅ It's false that p. ‡: The equally true and false exists. 0: The totally false exists ≅ The completely false is true. 1: The Truth exists (≅is true) ≅ Reality or Existence itself exists. p+q: (Either) p or q. p.q: p and q. pDq: The fact that p implies the fact that q ≅ It's at most as true that p as that q. Sp: It's both true and false that p ≅ It neither is nor fails to be the case that p. pCq: p only if q ≅ If p, (then) q. np: It's overtrue that p. mp: It's (very) much like true that p. Xp: It's very true that p. Lp: It's more or less true that p. Yp: It's infinitesimally true that p. fp: It's somewhat true that p. p‡q: It's less true that p than that q. Hp: It's completely (≅utterly≅altogether≅wholly≅totally) true that p. Fp: It's not at all (≅by no means) the case that p.

Now, let's come to *T*: *T* is the smallest subset of *F* which comprises every substitution-instance of any one of the following five axiom-schemata and that is

closed for the only rule belonging to R, to wit: *Modus ponens*, i.e.: pCq , p $\vdash$ q
AXIOM-SCHEMATA

In these axiom-schemata all schematic letters are unrestricted except in A2, wherein "r" is to differ from "s" at most in that $n(0 \leq n)$ occurrences of "q" lying in "s" are replaced in "r" by respective occurrences either of "p" or of "NNp".

A1 p.qCp   A2 pIqC.sIr
A3 p^(q.r)^sI(r^s.(q^s)^p)..p.qC(p^q)..p^qDp..pDq+Lp
A4 q+p.pIp..pIpI½..r.s+pI.r+p..s+p
A5 pIqC(qCp)..pDq+(qDnp+.pImq)..fSp.fSqC(p^q%.p.q)..
   XpDXqC(pDq)..mpDnp≡(Yp+YNp)..XmpImXp..mpDmnp+Hp.
   Hp+HqIH(p+q)..H(p.q)IL(Hp.Hq)..Y(p^q)C.Yp+Yq

The Tarski algebra of classes of formulas of Ap is formed by identifying $\boxed{p}$ with $\boxed{q}$ iff "pIq" is a theorem of Ap. The operations on that algebra are defined as is usual for other Tarski algebras of classes of formulas. Moreover, the Tarski algebra of classes of formulas of Ap is provably isomorphic with $F_2(\Gamma)$, Γ being the class of qq.tt.aa. So validity for Ap is very easy to establish: a valuation v of Ap is a mapping $F{\to}A$, where A is any q.t.a., provided: $v(a){=}m0$; $v(Hp){=}Hv(p)$; $v(p{+}q) = Nv(p).Nv(q)$; $v(p^q){=}v(p)^v(q)$; $v(pIq){=}v(p)Iv(q)$. A formula "p"ϵF is *valid* iff every valuation of Ap, v, is such that $v(p)$ ϵD, the respective set of dense elements of the algebra in which the range of v is included. (I suppose no ambiguity arises from our twofold use of symbols, as functors of Ap and as operations of any q.t.a.) Accordingly, Ap is both sound and complete.

§8.- TRANSITIVE ALGEBRAS PROPER

A *transitive algebra*, t.a. for short, is an algebra $<A,T> = A$ where $T = <1,B,N,H,n,+,\hat{},I>$ such that: 1º) $<A,<1,N,H,n,+,\hat{},I>>$ is a q.t.a.; 2º) B is a unary operation and the following postulate holds for every $x\epsilon A$:

(25) Either x+a=x=Bx; or else: a.x≠a and Bx=0.

Since the class of tt.aa., TA, is defined through an identity-disjunctive postulate, it's not closed for the formation of direct products; the free algebra with r≥2 generators associated with TA is *not* a a t.a., as we're going to see.

DEFINITION.- An *identity algebra* is an algebra $<A,<F,.,+,II>>$ such that: $<A,<F,.,+>>$ is a Stone algebra, and, for any x,y ϵA: F(xIIy)=0 iff x=y; and otherwise, xIIy=0.

Lemma 1.- Every t.a. is an identity algebra. Proof: let's definitionally introduce the operation II in any t.a. as follows: /xIIy/ eq /B(xIy)/. Then, applying Thm25, we know that a≤xIy, i.e. xIy+a=xIy iff xIy ϵD. But, in virtue of (25) a≤xIy entails that xIy=xIIy. In virtue of Corol of Thm10, xIy ϵD iff x=y; thus, if x=y then xIIy ϵD. On the other hand, if xIy ∉D, which in virtue of Corol of Thm10 happens iff x≠y, then, in virtue of Thm25, a≰xIy; i.e. xIy ≠ a; which in turn entails, in virtue of (25), that xIIy=0. Therefore: xIIy=½ iff x=y; and xIIy=0 iff x≠y. That completes the proof, since F½=0.

Lemma 2.- Every identity algebra is simple, which means that it has only two congruences ι and ω. Proof: suppose xθy. Then (xIIx)θ(xIIy). Since xIIx ϵD, then, if xIIy=0, θ=ι (in virtue of a well-known fact about Stone algebras, viz that the only congruence θ such that, for at least some dense element x, xθ0 is the universal congruence ι, since, if x is dense and xθ0, then FFxθFF0, i.e. 1θ0); hence, if (xIIx)θ(xIIy) and θ≠ι, then xIIy≠0, so x=y; there-

fore, if xθy and x≠y, θ=ι.
Theorem 36.- Every t.a. is simple.
1st Corollary: Every t.a. is subdirectly irreducible. Proof: cf. Theorem I/3 of Balbes & Dwinger.
2d Corollary.- Any morphism from a t.a. into another is an embedding. Therefore, every epimorphism of a t.a. onto another is an isomorphism.
Theorem 37.- The free algebra with r≥2 generators associated with TA is not a t.a. Proof: The only identities that hold in that free algebra, $F_r(TA)$, are the ones that hold in all and every t.a. Hence, for some polynomials, p and q, such that in some but not in all tt.aa. p=q, it is in $F_r(TA)$ the case that p≠q; therefore, were $F_r(TA)$ a t.a., pIIq=0. But that is impossible, since then some equation would hold in $F_r(TA)$ that did not hold in every t.a. Consequently, $F_r(TA)$ ∉ TA.
Theorem 38.- Every scalar q.t.a. is a t.a. wherein Bx is trivially "defined" like this: /Bx/ eq /x/. And, conversely, a t.a. such that, for every x, Bx=x, is a scalar q.t.a. Proof of the first part: (25), Thm32&35. Proof of the second part: in such an algebra xIy=xIIy; but, for any x and y, xIy = either ½ or 0. Then we apply Thm32.
1st Corollary: Every direct product $\Pi t\epsilon T(A_t)$ of a family (A_t) a scalar qq.tt.aa. can be made into a t.a. by adding thereto an additional operation B as follows: Bx=x if, for every projection-function p_t, $p_t(x)$ is a dense element of A_t; and otherwise, Bx=0.
2d Corollary: Let A be a t.a. and let A′ be a sub-algebra of A whose carrier, $A′$, is such that, for every $x\epsilon A′$, x=Bx. Then, A′ is a scalar q.t.a. (Notice, though, that that by no means entails that a t.a. whose set of generators is such that, within that set, x=Bx for every x is a scalar q.t.a. For, even if x=Bx and y=By, it may be the case that xIy ≠ xIIy.)

THE SYSTEM OF TRANSITIVE LOGIC Aj

Aj is Ap adding to its vocabulary the one-place functor 'B'; and adding one additional axiom-schema and an additional inference-rule, the assertion-rule, viz: p $\vdash$ Bp. The additional axiom-schema is:
A6 BLpDLBp..Bp+BFBp..BpIp+FBp..B(pDq)C.BpDBq

A valuation of Aj will be a mapping, v, of the set of formulas of Aj into any t.a., A, which is like a valuation of Ap and, besides, is such that $v(Bp) = Bv(p)$. Of course, a formula "p" of Aj is valid iff every valuation, v, of Aj is such that $v(p)$ ϵD. With that semantics, Aj is both sound and complete.

§9.- REDUCTIVE QQ.TT.AA.

I call *reductive* any q.t.a. wherein some further equations hold containing 'K' and 'X'. A q.t.a. is *strongly reductive* iff, for any x thereof, KXx=x. Consequently, in any strongly reductive q.t.a., KXx=x=XKx. $\tilde{Z}$ is a strongly reductive t.a. ($\tilde{A}$ is a reductive t.a., too, since, for any $x\epsilon\tilde{A}$, XXKx≤KXx, and KXx≤XKx.) We can set up a *reductive transitive logic* by adding to its axioms this one:
A7 KXpIp (where /Kp/ eq /NXNp/)

My own feeling is that (every instance of) A7 is true. The reader is invited to think it out himself.

Invited Address

PREVIOUS PAGE
IS BLANK

THE IMPLEMENTATION AND USE OF MULTIVALUED
LOGIC IN A VLSI ENVIRONMENT

H. Fleisher
IBM Fellow

IBM Corporation
D/C14, B/704, P. O. Box 390
Poughkeepsie, New York 12602

ABSTRACT

In this paper, multiple valued logic is discussed in terms of its implementation by PLAs. The history of the PLA is described and the relevance of multivalued logic to PLA development and logic minimization is discussed. An important aspect of VLSI implementation is used as a parameter to compare binary logic circuits with multiple valued logic circuits and certain conclusions are reached. The functional usefulness of n-valued Post logics is identified in terms of various applications.

HISTORY OF PLA AND
ITS RELATION TO MULTIVALUED LOGIC

The PLA emerged in the late 1960s and early 1970s as an approach to utilizing more effectively the attributes of LSI. It had been well established that regular structures such as RAM and ROM were more effectively mapped at high density into LSI than random logic. Knowing this, I started an investigation into the use of memory-like structures into which logic could be effectively mapped.

However we recognized that the typical PLA structure, while avoiding the exponential growth of memory cells of a conventionally addressed memory when used to implement combinatorial logic, nevertheless also had an exponential growth factor built into it when the input bits addressed the PLA via 1-bit decoders, in a manner analogous to addressing an associative memory. In the effort to develop a PLA which would contain, to a greater extent, the rapid growth in memory cells that occurs with either the conventionally addressed memory structure or the associatively addressed PLA structure as the number of variables increases, a method of partitioning, or grouping, the input variables was devised [1], [2]. Each

group, in effect, becomes a "super binary variable", and the functions to be mapped, are first transformed into groupings of sub-functions. As shown in Figure 1, this grouping (mathematically identical with factoring) can result in a substantial reduction in the number of PLA cells required to implement the function, in this example, the symmetric exclusive-OR.

If we group two binary variables into a single "super binary variable", we generate a variable which has four values associated with it. A grouping of three binary variables results in a single variable with eight values, and so on. This enumeration describes a subset of n valued Post algebras, containing only those n that can be expressed as a power of two.

So, it is clear that this type of multivalued logic has been important in developing the bit-partitioned PLA structure. Hong, Ostapko, and Cain [3] recognized this aspect of bit-partitioning in their MINI program, which heuristically determines the minimum number of cells needed to implement functions under different groupings of input variables. Sasao [4] also made use of MINI to manipulate multivalued logic in using PLAs to implement multivalued functions.

THE USE OF PLAS TO IMPLEMENT
MULTIVALUED LOGIC FUNCTIONS

The implementation of multiple valued logic by appropriate hardware has received extensive treatment, especially efforts to devise circuits to handle multivalued logic as extensions of binary circuits. N-valued logic circuits must have n stable states and the transmitted signals must have n distinguishable values. Although PLAs, as presently constituted, are binary output devices, nevertheless, in light of the discussion above, they can readily be used to implement n-valued Post logics. With appropriate coding, they can also be used

138

to implement other logics as well. As indicated above, a single variable of n Values can be encoded by a group of binary valued variables containing k bits where k is the smallest integer such that 2**k n. Thus for n=3, k=2; for n=4, k=2. For n=5 through n=7, k=3, etc. With this encoding in mind, we shall discuss three functions in n-valued Post logic: the MAX, MIN and arithmetic (SUM/CARRY) functions.

These are illustrated for n=3, 4 in Figure 2.

It is clear from these tables, that the MAX and MIN functions reduce to OR and AND for n=2, the binary case, and that we have defined the SUM and CARRY functions for single digit addition with no carry in. Using positional binary coding, we present these functions in Figure 3a.

As is well known, the configurations shown in Figure 3 can be directly translated into PLAs with the input bits grouped as (X1, X2) and (Y1, Y2).

However, the proper grouping of bits can result in a substantial reduction in the number of cells needed to implement the function [2]. In the case of the SUM function, a grouping of (X1, Y1) and (X2, Y2) produces a more efficient mapping.

Figure 4 shows a tabulation and a plot of the number of products versus radix for a one stage adder with no carry in. This tabulation was accomplished using IDL [5] to design the adders in the different radices. MINI [6], as incorporated in IDL, was then used to obtain the optimum PLA configuration for each radix, using the "vertical" bit pairing shown above.

The graph shows great variation in efficiency of mapping these arithmetic functions. Strong minima are shown for radices 8, 16, and 32, all of which are powers of two. Other strong minima are shown for radices 12 and 24, both containing 3 as a factor multiplying powers of two.

A significant aspect of mapping multivalued logic functions into PLAs by means of encoding the variables and outputs from the given radix into binary groups is that a standard hardware is used for all radices. In this way, effective comparison can be made of relative hardware utilization.

COMPARISON OF BINARY AND MULTI-VALUED LOGIC CIRCUITS

As we have discussed, binary encoded multivalued logic may be mapped into binary switching circuits. However multivalued logic may also be directly mapped into multistate circuits. There are many papers in the literature describing efforts to devise multivalue logic circuits that will send multilevel signals from one logic circuit to others, making more efficient use of the interconnecting wiring.

K. C. Smith, in a recent paper [7], relates advances in multiple valued logic to advances in integrated circuit technology. While he recognizes possible advantages to multivalued logic for encoding information, he neglects the trade offs involved in considering such parameters as power dissipation and relative area of active elements versus area needed for wiring. Thus it is instructive to analyze these circuits from the point of view of power dissipation and area on the silicon chip [8]. For comparison, I will use a binary switching circuit, an n valued switching circuit, and a binary encoded n valued switching circuit, all represented in idealized form. These are shown in Figure 5.

In each of these circuits, the capacitor, C. represents the combined output capacity of the sending switching circuit and the input capacity of the receiving switching circuit. Transmission line effects are neglected as are fan-in and fan-out considerations. We also assume the value of the capacitor to be the same for each circuit configuration.

In Figure 5a, the signal on the line is either O or Vout, corresponding to the logic values of 0 or 1.

The energy to be switched is therefore the energy stored in the capacitor:

$$E_{bsw} = CV_{out}^2/2$$

In Figure 5b, the signal on the line ranges in discrete values: 0, Vout, 2Vout, ... (n-1) Vout corresponding to the logic values 0, 1, 2, ... (n-1) in the n valued logic.

Therefore the maximum energy to be switched corresponds to the maximum logic value (n-1):

$$E_n^{max} = (n-1)^2 V_{out}^2 C/2 = (n-1)^2 E_{bsw}$$

Figure 5c shows the k lines for binary encoding of n valued logic. In this case, the maximum energy to be switched simultaneously is that of the k capacitors each charged to Vout, yielding

$$E_k^{max} = kCV_{out}^2/2 = kE_{bsw}$$

circuit.

Hence a fair analysis requires a discussion of the physical parameters as well as the logical factors.

To analyze the silicon chip area associated with circuitry, let the minimum area associated with the binary switch be denoted by A_B. Then the energy density for the binary switch is

$$E_{bsw}/A_B$$

For the n valued logic circuit, the area must be determined to maintain the same energy density:

$$E_n^{max}/A_n = E_{bsw}/A_B$$

whence:

$$A_n = (n-1)^2 A_B$$

Similarly, we find that the area associated with k binary circuits is kA_B.

It would appear that the n**2 growth of both energy (power dissipation) and area of the circuit needed to handle the power dissipation militate against the use of single line multivalue logic circuits, but the linear growth of energy and area for binary encoded n valued logic consumption implementations may still permit the use of such logics where system design considerations and applications may be advantageous. Thus, for ternary logic, n = 3, and k = 4, and $A_n = 4A_B$ while $A_k = 2A_B$.

APPLICATIONS

The possible application of multivalued logic to arithmetic operations is fairly straight forward, and has been extensively discussed in the literature. For example, V. C. Hamacher and Z. G. Vranesic [9], in an analysis of ternary logic applied to multipliers conclude that the use of ternary gates will result in a 70% reduction in gate cost and an 80% reduction in input count, with, however, a doubling of delay. The cost factor is somewhat ambiguous however, since they assign the same costs to the ternary primitives of AND, OR, and INVERT as are assigned to the same binary primitives. Moreover, as they state, such factors as power consumption, level of integration, etc. were not taken into account in their discussion. As we determined in the previous section, power consumption of n-valued logic circuits transmitting multiple signal levels on a single line have a power consumption of the order of n**2 times the power consumption of the comparable binary

A more recent example of a possible application of multivalued logic appears in "A Unified Switching Theory With Applications to VLSI Design" [10] by John P. Hayes. Hayes develops a theory that attempts to unify classic switching theory with circuit parameters. In so doing, he introduces 4 basic types of logic values - the Boolean 0 and 1, an indeterminate value, U, and the high impedance state of the circuit, Z. These values can be treated as a 4 valued logic, and appear to be relevant in analyzing and testing circuits.

An interesting relationship evolved by the author, is the similarity between the binary logic circuits as evolved by his theory and relay networks on the one hand, and PLAs on the other.

A third application of multivalue logic arises in handling some of the procedural aspects of programming. For example, radix conversion must be programmed for many commercial applications, for data entry into the system, reformatting and processing, and for exit of the processed data.

Similarly, the use of multivalue logic may contribute to simplifying instruction sets.

While these are fairly simple examples, they are, nevertheless, suggestive of a growing role MVL may play in a world dominated by binary-value hardware.

SUMMARY

Post n-value algebras are currently in use in aiding logic design of computer functions. However their direct implementation by means of n-value circuits does not appear to be promising because of higher power dissipation and area requirements than binary logic, as well as other factors. On the other hand, binary encoded multivalue logic circuits retain many of the advantages of binary circuits. These will enable the designer to make optimum use of multivalue logic.

ACKNOWLEDGMENT

REFERENCES

[1] H. Fleisher, A. Weinberger, V. Winkler, "The Writeable Personalized Chip," Computer Design, Vol. 9, No. 6, pp. 59, 1970, and H. Fleisher, A. Weinberger, V. Winkler, "Partitioning Logic Operations in a Generalized Matrix System," U.S. Patent #3,593,317, July 13, 1971.

[2] H. Fleisher, L. Maissel, "An Introduction to Array Logic," IBM Journal Research and Development, Vol. 19, No. 2, March 1975, pp. 98-109.

[3] S. J. Hong, D. L. Ostapko, R. G. Cain, "A Practical Approach to Two-Level Minimization of Multivalued Logic", Proceedings 1974 Intern Symposium on Multiple-Valued Logic, West Virginia University, Morgantown, W. Va.

[4] T. Sasao, "Multiple Valued Decomposition of Generalized Boolean Functions and the Complexity of Programmable Logic Arrays", IEEE Transactions on Computers, Vol. C-30, No. 9, September 1981, pp. 635-643.

[5] L. I. Maissel, D. L. Ostapko, "Interactive Design Language: A Unified Approach to Hardware Simulation, Synthesis and Documentation", ACM, IEEE, Nineteenth Design Automation Conference Proceedings, ISSN 0146-7123, paper 14-1, pp. 193-201.

[6] S. J. Hong, D. L. Ostapko, R. G. Cain, "MINI - A Heuristic Approach to Logic Minimization", IBM Journal of Research and Development, Vol. 18, 1974, pp. 443-448.

[7] K. C. Smith, "The Prospects for Multivalued Logic: A Technology and Applications View", IEEE Transactions on Computers, Vol. C-30, No. 9, September 1981, pp. 619-634.

[8] R. W. Keyes, "Physical Limits in Digital Electronics", Proceedings of the IEEE, Vol. 63, No. 5, May 1975, pp. 740-767, and ..., "Fundamental Limits in Digital Information Processing", Proceedings of the IEEE, Vol. 69, No. 2, February 1981, pp. 267-278.

[9] V. C. Hamacher, Z. G. Vranesic, "Multivalued Logic in Arithmetic Units", Chapter 17, "Computer Science and Multiple Valued Logic", ed. D. C. Rine, North Holland Publ. Co., 1977, pp. 485-505.

[10] John P. Hayes, "A Unified Switching Theory With Application to VLSI Design", Proceedings IEEE, Vol. 70, No. 10, October 1982, pp. 1140-1151.
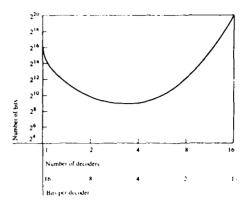
Figure 1 Number of bits needed to implement the EXCLUSIVE OR of 16 variables as a function of number of bits per decoder

(a) n=3

MAX - 0 1 2
0 - 0 1 2
1 - 1 1 2
2 - 2 2 2
-

MIN - 0 1 2
0 - 0 0 0
1 - 0 1 1
2 - 0 1 2
-

SUM - 0 1 2
0 - 0 1 2
1 - 1 2 0
2 - 2 0 1
-

CARRY - 0 1 2
0 - 0 0 0
1 - 0 0 1
2 - 0 1 1
-

(b) n=4

MAX - 0 1 2 3
0 - 0 1 2 3
1 - 1 1 2 3
2 - 2 2 2 3
3 - 3 3 3 3
-

MIN - 0 1 2 3
0 - 0 0 0 0
1 - 0 1 1 1
2 - 0 1 2 2
3 - 0 1 2 3
-

SUM - 0 1 2 3
0 - 0 1 2 3
1 - 1 2 3 0
2 - 2 3 0 1
3 - 3 0 1 2
-

CARRY - 0 1 2 3
0 - 0 0 0 0
1 - 0 0 0 1
2 - 0 0 1 1
3 - 0 1 1 1
-

MAX, MIN, SUM, CARRY FUNCTIONS FOR n=3, n=4

FIGURE 2

BINARY ENCODED TABLES FOR MAX, MIN, SUM, CARRY FUNCTIONS FOR n=3

FIGURE 3a

RADIX	PRODUCT TERMS				
2	2	13	37	24	27
3	4	14	29	25	73
4	5	15	27	26	63
5	10	16	17	27	70
6	9	17	44	28	53
7	15	18	44	29	75
8	10	19	55	30	61
9	23	20	39	31	43
10	21	21	64	32	26
11	26	22	52	33	78
12	17	23	43		

FIGURE 4a

BINARY ENCODED TABLES FOR MAX, MIN, SUM, CARRY FUNCTIONS FOR n=4

FIGURE 3b
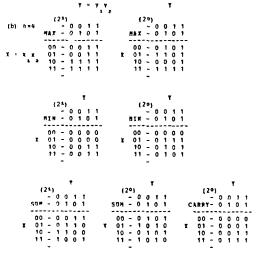
RADIX VS. PRODUCT TERMS

ONE STAGE OF ADDER

(VERTICAL PAIRS / COMPLEMENTED OUTPUTS)

Figure 4b

RADIX
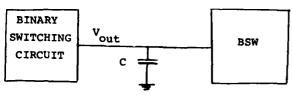
FIGURE 5a
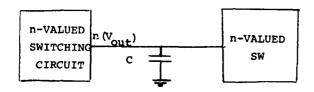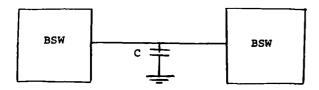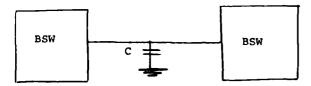
FIGURE 5b

k BSW circuits
for
binary encoded
n value logic
$k=\log_2 n$

FIGURE 5c

143

Session 4A
Charge-Coupled Devices and Applications

PREVIOUS PAGE
IS BLANK

PULSE-TRAIN RESIDUE ARITHMETIC CIRCUIT USING MULTIPLE-VALUED CHARGE-COUPLED DEVICES
AND ITS APPLICATION TO DIGITAL FILTER

Nobuhiro TOMABECHI[+], Michitaka KAMEYAMA[++] and Tatsuo HIGUCHI[++]

+ Department of Electrical Engineering, Faculty of Engineering,
  Hachinohe Institute of Technology, Obiraki, Myo, Hachinohe, 031 Japan

++ Department of Electronic Engineering, Faculty of Engineering,
   Tohoku University, Aoba, Aramaki, Sendai, 980 Japan

ABSTRACT

A new design method of the compact residue arithmetic circuit using multiple-valued charge-coupled devices (CCD's) is proposed. The multiple-valued ring counter for the residue arithmetic is designed by using the CCD's. Because the structure of the counter is very simple, it is effectively used as the basic component to construct the residue arithmetic circuit. The modulo-m addition is performed by shifting the modulo-m multiple-valued ring counter, and the coefficient multiplication is done by converting the multiple-valued code between the counters. The most important advantages of the proposed adder and multiplier are the compact hardware and the uniform operating time, so that these arithmetic circuits can be effectively employed for the pipelining digital signal processing system. Finally, it is demonstrated that the hardware complexity of the digital filter constructed with the quaternary logic CCD's can be reduced to 70% compared with the corresponding binary implementation.

I. INTRODUCTION

In the residue number system, the operation of addition and multiplication can be performed very quickly because of the separability of operations on each digit [1]. These residue arithmetic circuits are usually implemented by storing arithmetic tables in read only memories (ROM's) [2-4]. Much memory is required in the implementation using ROM's for the storage of the arithmetic tables so that the complexity of the hardware is increased.

On the other hand, ring counters are suitable for residue arithmetic because of its inherent nature of circulation. We have presented a new residue arithmetic circuit called pulse-train residue arithmetic circuit by the use of conventional ring counters [5-6]. It has been made clear that the number of gates and memory celles in the pulse-train residue arithmetic circuit can be reduced to 1/100 of that for the equivalent ROM.

Charge-coupled devices (CCD's) can offer very simple charge storage and transfer with very dense LSI or VLSI structures [7-8]. Multiple-valued charge storage can be done in the CCD because it is an analog device by ature. By using these advantages, multiple-valued memories and multiple-valued

logic implementation by CCD's have been investigated [9-10].

In this paper a new design method of compact residue arithmetic circuits using multiple-valued CCD's is proposed. A multiple-valued ring counter for residue arithmetic circuits using CCD's is presented. The multiple-valued ring counter is essentially an N-stages feedback shift register operating such that only one of the memory elements takes the logical values except 0. Some types of the multiple-valued ring counter have been designed [11-12]. In this paper, the implementation of a multiple-valued ring counter with compact hardware is considered, because it can be easily implemented by CCD's. By the use of multiple-valued CCD's the number of cascade chains of the memory elements in the counter can be significantly reduced. The multiple-valued ring counter is used as the basic component to construct the residue arithmetic circuit.

The residue adder and the residue coefficient multiplier using the multiple-valued CCD ring counter as a main component is also proposed. The modulo-m addition is performed by means of shifting the modulo-m multiple-valued ring counter. On the other hand, the modulo-m coefficient multiplication is realized by the multiple-valued code conversion between the multiplicand counter and the product counter. These residue adder and the residue multiplier can be implemented with the very compact hardware, and the operations can be performed in the uniform operating time, so that they can be effectively used as the basic building block for the pipelining digital signal processing systems. As an example, an n-th order non-recursive digital filter is designed. It is made clear that the hardware complexity of the digital filter constructed by quaternary logic CCD's is reduced to 70% compared with the corresponding binary implementation.

II. OVERVIEW OF THE RESIDUE NUMBER SYSTEM

In the residue number system (RNS), an integer X is represented by N-tuples as

$$X = (x_0, x_1, \cdots, x_{N-1}) \qquad (1)$$

where x_i is the remainder of X divided by the i-th modulus m_i, and is denoted by

146

$$x_i = |X|_{m_i} \tag{2}$$

If all values of m_i are mutually prime integers, the dynamic range of X is

$$0 \le X \le M - 1 \tag{3}$$

where $M = \prod_{i=0}^{N-1} m_i$. The addition and the multiplication can be done separately in each digit, which are given by

$$X + Y = (x_0 \oplus y_0, \ x_1 \oplus y_1, \ \ldots, \ x_{N-1} \oplus y_{N-1}) \tag{4}$$

$$X \cdot Y = (x_0 \odot y_0, \ x_1 \odot y_1, \ \ldots, \ x_{N-1} \odot y_{N-1}) \tag{5}$$

where $\oplus$ and $\odot$ are modulo-m_i addition and multiplication on each modulus, respectively, which are written by

$$x_i \oplus y_i = |x_i + y_i|_{m_i} \tag{6}$$

$$x_i \odot y_i = |x_i \cdot y_i|_{m_i}. \tag{7}$$

III. MULTIPLE-VALUED RING COUNTER USING CCD'S

3.1 Logic model for CCD's

Let the set of the logic value in the r-valued logic system be $L = \{0, 1, \ldots, r-1\}$ and $p = r-1$. A CCD acts as a shift register storing analog or multiple-valued logic informations. Let us consider the r-valued logic CCD's including binary logic. A charge packet Q of the CCD can be written as

$$Q = nQ_E \quad (n = 0, 1, \ldots, r-1) \tag{8}$$

where Q_E is a unity charge packet.

The set of logic models for CCD's proposed by Kerkhoff [13] will be used in the following system design, because this set of logic operators is functionally complete, namely any multiple-valued logic function can be implemented by using the logic operators. In Fig. 1 the set of logic operators is shown, which consists of addition, constant, fixed overflow and inhibit. A simple delay element is included in Fig. 1 as a logic operator, because it is most frequently used in the counter. The detailed explanation of these operators is found in Reference [13]. Signals in the logic operators is "charge", whereby there are some restrictions such that fan-outs of a signal and crossing of signal lines should be permitted only at the places where the logic level can be regenerated. The common clock pulse is fed to each operators so that logic operations are performed with synchronizing to the clock pulse in the pipelining manner.

In the following discussion, let us introduce "clock control". By the use of clock control, the clock pulse applied to the operator can be controlled by the output of the other operators. By means of the clock control the counter can be easily realized by the CCD shift register. Furthermore, the single output of each operator can be fed to the input of the other operators controlled by the different clock pulses, namely many fan-outs of each operator become available.

3.2 Implementation of the multiple-valued ring counter

A multiple-valued ring counter is composed of N-stages of multiple-valued memory elements cascaded in the feedback shift register form as shown in Fig. 2 (a). For the modulo-m arithmetic operation the number of stages N is given by

$$N = \lfloor m/p \rfloor \tag{9}$$

where $\lfloor x \rfloor$ denotes the smallest integer such that $\lfloor x \rfloor \ge x$. The states of the memory elements are assigned such that only one memory element stores the value except 0. For an example, the state assignment of the ternary counter in modulo-7 is shown in Fig. 2 (b). Let the state of the counter $Q = (q_0, q_1, \ldots, q_{N-1})$, and let q_x take the value $x + 1$ except 0, where $x + 1 \in L$. Then, the number stored in the counter is expressed by

$$Y = N \cdot x + X. \tag{10}$$

Symbol	Circuit	Logic function
Addition		$Q_o = \sum_{i=1}^{n} Q_{im}$ $n=2,3,\ldots$
Constant		$Q_o = L$ $L=1,2,\ldots$
Fixed Overflow		
Inhibit		
Delay		$Q_o = Q_i$

Fig. 1 CCD logic operators proposed by Kerkhoff

147

The following notation is also used for the expression of Y:

$$Y = (x, X). \tag{11}$$

The waveform of the input and output signals of the counter is shown in Fig. 2 (c), which is expressed by

$$A(kT) = \begin{cases} 0 & \text{if } k \neq X \\ 1 & \text{if } k = X \end{cases} \tag{12}$$

where $A(kT)$ is the input and output at the time kT, and X is the number expressed by the signal.

The shift operation of the counter is given by

$$q_i' = q_{i-1}$$

$$q_0' = \begin{cases} q_{N-1} & \text{if } q_{N-1} = 0 \\ q_{N-1} + 1 & \text{if } q_{N-1} \neq 0 \end{cases} \tag{13}$$

where q_i' represents the next state of q_i. The excitation of q_0' is performed by the feedback gate FG in Fig. 2 (a). The counter must be reset to the initial state $Q_0 = (1, 0, \ldots, 0)$ when the content in the counter becomes m as well as at the initial timing. This reset condition is expressed by

$$q_x = p$$

$$x = (p + 1)N - m. \tag{14}$$

A ternary ring counter constructed with CCD's is shown in Fig. 3 (a). The operation of the counter is illustrated in Fig. 3 (b). First of all, the counter is reset to the initial state $(1, 0, \ldots, 0)$ by the clock pulse ϕ_p. Next, shifting is performed by the clock pulse ϕ_f until the arrival of the signal $A(kt)$. The feedback gate FG and the reset gate RG can be implemented using CCD's as shown in Fig. 4.

To construct the shift control circuit MOS devices may be preferable, since the circuit needs power to drive and MOS devices has good compatibility with a CCD.
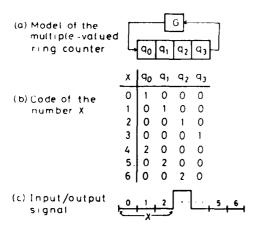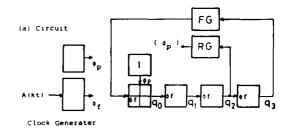
In the multiple-valued ring counter proposed here, the number of delay elements can be greatly reduced compared with the corresponding binary ones.

IV. RESIDUE ADDER AND RESIDUE COEFFICIENT MULTIPLIER

Consider the design of an adder and a coefficient multiplier on each digit using the multiple-valued ring counters as main components.

Fig. 5 shows the residue adder which consists of a difference counter and a buffer counter. The difference counter is designed as a bilateral shift register so as to perform the subtraction, A − B. The operation of the adder is shown in Fig. 6 in the case of A = 5, B = 2 and m = 7, namely the case of A ≥ B. First, the difference counter is reset by the clock pulse ϕ_p. From input signals HA and HB are produced, respectively. The difference,
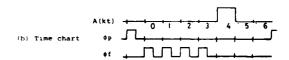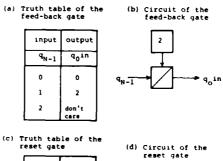

(a) Circuit

Clock Generater

(b) Time chart

A(kt)

ϕ_p

ϕ_f

Fig. 3 Multiple-valued ring counter composed of CCD's

(a) Truth table of the feed-back gate

input	output
q_{N-1}	$q_0$in
0	0
1	2
2	don't care

(b) Circuit of the feed-back gate

(c) Truth table of the reset gate

input	output
q_{N-2}	rp
0	0
1	0
2	1

(d) Circuit of the reset gate

Fig. 4 Feedback gate and the reset gate

(a) Model of the multiple-valued ring counter

(b) Code of the number X

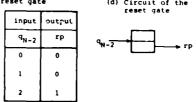| X	q_0	q_1	q_2	q_3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	2	0	0	0
5	0	2	0	0
6	0	0	2	0

(c) Input/output signal

Fig. 2 Multiple-valued ring counter

A-B can be obtained in the counter by applying the right shift pulse ϕ_f in the period of HA = 1 and HB = 0. The content in the difference counter is transfered to the buffer counter at the timing ϕ_p. Then, the content in the buffer counter is transmitted to the next stage in the next operating cycle. Fig. 7 shows the operation in the case of A < B, in which A - B is obtained in the form of the complement by applying the left shift clock pulse ϕ_r in the period of HA = 0 and HB = 1, where the complement, $\bar{X}$ of a number X is defined by

$$\bar{X} = m - X \qquad (15)$$

By adapting subtraction as the basic operation and by the use of the buffer counter, the operation of the addition can be finished in the time given by

$$T = (m + 1) \tau \qquad (16)$$

where τ represents the interval between each clock pulse. The operation of the adder is expressed by

$$X(kT) = \left| A((k-1)T) - B((k-1)T) \right|_m \qquad (17)$$

where X(kT) denotes the output at the operating cycle of kT, and A((k-1)T) and B((k-1)T) denote the inputs at the previous cycle of kT.

On the other hand, the coefficient multiplication can be done by the multiple-valued code conversion between the multiplicand counter and the product counter. The code conversion is performed by

$$(y, Y) = N \cdot y + Y$$

$$= \left| K \cdot X \right|_m \qquad (18)$$

where X and (y, Y) denote the multiplicand and the product, respectively. Eq. (18) implies that the value of y + 1 is set to the Y-th memory element of the product counter. Table 1 shows the multiplication table in the case of m = 7, K = 3. Fig. 8 shows the multiplier following the Table 1, which consists of the multiplicand counter, the product counter, the line exchanger and the level converter. The code conversion is realized by the line exchanger and the level converter. The code conversion is done at the final timing ϕ_p after the multiplicand A is set to the multiplicand counter. The content in the product counter is used in the next operating cycle.

Since the voltage mode operation can be done in the control line of the inhibit operator [10], the crossing over of the signal lines can be possible in the line exchanger.

The operating time of the multiplier is given by Eq. (16) similarly as in the case of the adder. The function of the multiplier is represented by

$$X(kT) = \left| K \cdot A((k-1)T) \right|_m \qquad (19)$$

where A((k-1)T) and X(kT) denote the input and the output, respectively.

The adder and the multiplier presented here are very compact and regularly constructed with uniform delay time, so that they can be effectively employed as the basic building blocks for the pipelining digital signal processing system.
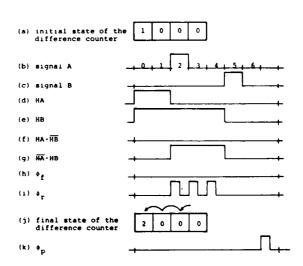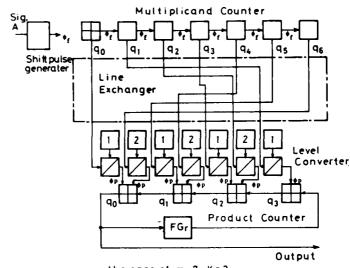

Fig. 5 Adder

Fig. 6 Operation of the adder A = 5, B = 2

Fig. 7 Operation of the adder A = 2, B = 5

Table 1 Multiplication table
the case of K = 3, m = 7

		Product			
		q_0	q_1	q_2	q_3
Multiplicand A	0	1	0	0	0
	1	0	0	0	1
	2	0	0	2	0
	3	0	0	1	0
	4	0	2	0	0
	5	0	1	0.	0
	6	2	0	0	0

the case of m=7, K=3

Fig. 8 Multiplier

V. APPLICATION TO DIGITAL FILTER

5.1 Digital filter realization

In this section the implementation of a digital filter using the residue adder and the residue coefficient multiplier is discussed. Consider the implementation of the n-th order non-recursive digital filter which is represented by

$$y(kT) = x(kT) + b_1 x((k-1)T)$$
$$+ b_2 x((k-2)T)$$
$$+ \ldots\ldots\ldots$$
$$+ b_n x((k-n)T) \qquad (20)$$

where x(kT) and y(kT) are the input and the output of the digital filter at the time kT, respectively. According to Eq. (20) the digital filter can be realized as shown in Fig. 9. The fractional coefficients such as b_1, b_2, ... , and b_n are transformed into the integer by multiplying a constant K, because the residue number system defined over the integer. The delay function is realized in the adders and multipliers, so that none of delay elements are used in the digital filter.

The operation of the digital filter can be performed in the pipelining manner because all of the adders and multipliers operate in the same operating time. The sampling time of the digital filter takes only one operating time , i.e., T, so that the operating speed of the digital filter will be increased.

Since the basic building blocks used in the digital filter are very compact and CCD's offer very dense LSI or VLSI structure, the digital filter may be completely implemented on a single LSI chip

Fig. 9 shows the digital filter in each modulus. The dynamic range of the digital filter, M is

determined by the product of all the moduli given by Eq. (3). Because of the separable nature of the residue arithmetic it is feasible to completely separate a digital filter to the sub-filters corresponding to each modulus m_i. Therefore, the structure of the digital filter becomes significantly regular.

5.2 Evaluation of the hardware complexity

In Table 2, the numbers of operators required in the adder, the multiplier and the 2nd order non-recursive digital filter with one modulus are shown in the cases using binary, ternary and quaternary logic CCD's. Only the number of main logic operators constructing the system, which depends on the modulus m_i is considered. In order to evaluate the hardware complexity, the operator must be weighted by the relative complexity factors. In

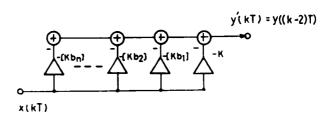$$y(kT) = y((k-2)T)$$

$$x(kT)$$

Fig. 9 n-th order non-recursive digital filter

Reference [13], the relative cost factors have been also described, which are shown in Table 3. Let us take these factors as the relative complexity factors for the logic operators. The weighted sum of the operators is shown in the right-most column of Table 2, it is found that the hardware complexity of the digital filter constructed by the quaternary logic CCD's can be reduced to 70% compared with the corresponding binary implementation.

VI. CONCLUSION

In this paper, a design method of compact residue arithmetic circuits suitable for multiple-valued CCD implementation has been presented. These residue arithmetic circuits can be effectively employed for the pipelining digital signal processing system, because of the reduction of the hardware and the uniform operating time. An application to n-th order non-recursive digital filter is demonstrated. It is made clear that the hardware complexity of the digital filter constructed by quaternary logic CCD's can be reduced to 70% compared with the corresponding binary implementation.

The detailed design of the residue arithmetic circuits using CCD's remains as a future problem.

REFERENCES

[1] N. S. Szabo and R. I. Tanaka, Residue Arithmetic and Its Application to Computer Technology, McGraw-Hill, 1967.

[2] W. K. Jenkins and B. J. Leon, "The Use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters," IEEE Trans. Circuits Syst., vol. CAS-24, pp. 191-201, Apr. 1977.

[3] G. A. Jullien, "Residue Number Scaling and Other Operations Using ROM Arrays," IEEE Trans. Comput., vol. C-27, pp. 325-336, Apr. 1978.

[4] W. K. Jenkins, "Recent Advances in Residue Number Techniques for Recursive Digital Filtering," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-27, pp. 19-32, Feb. 1979.

[5] N. Tomabechi, "Residue Arithmetic Using Ring Counters and Its Error Correcting Circuit," IECE Japan Trans., vol. J64-D, pp. 545-546, June, 1981.

[6] N. Tomabechi, M. Kameyama and T. Higuchi, "Pules Rate Arithmetic Circuit Based on Residue Number System and Its Application to Digital Filter," IECE Japan Trans., vol. J65-D, Feb. 1982.

[7] C. H. Squin and M. F. Tompsett, Charge Transfer Devices, Academic Press, 1975.

[8] T. A. Zimmerman, R. A. Allen and R. W. Jacobs, "Digital Charge-Coupled Logic (DCCL)," IEEE J. Solid-State Circuits, vol. SC-12, pp. 473-585, Oct. 1977.

[9] M. Yamada, K. Fujita, K. Nagasawa and Y. Gamou, "A New Multilevel Storage Structure for High Density CCD Memory," IEEE J. Solid-State Circuits, vol. SC-13, pp. 688-692. Oct. 1978.

[10] H. Kerkhoff and M. L. Tervoert, "Multiple-Valued Logic Charge-Coupled Devices," IEEE Trans. Comput., vol. C-30, pp. 644-652, Sept. 1981.

[11] T. Higuchi and M. Kameyama, "Static Hazard-Free T-Gate for Ternary Memory Element and Its Application to Ternary Counters," IEEE Trans. Comput., vol. C-26, pp. 1212-1221, Dec. 1977.

[12] N. Tomabechi, M. Kameyama and T. Higuchi, "Efficient Residue Arithmetic Circuit Using Multiple-Valued Ring Counters and Its Application to Digital Signal Processing," Proc. of Int. Symp. on Multiple-Valued Logic, pp. 107-112, May 1982.

[13] H. Kerkhoff and H. A. J. Robroek, "The Logic Design of Multiple-Valued Logic Functions Using Charge-Coupled Devices," Proc. of Int. Symp. on Multiple-Valued Logic, pp. 35-44, May 1982.

Table 2 Comparison of the hardware complexity

	Adder	Multiplier				
	Addition	Addition	Constant	Fixed overflow	Inhibit	Delay
Binary	2m	m	m	-	m	m
Ternary	m	.5 m	m	-	m	m
Quaternary	.66m	.33m	m	-	m	m

	Digital filter					
	Addition	Constant	Inhibit	Delay	Total	Weighted Sum
Binary	9m	3m	3m	3m	18m	42m
Ternary	4.5m	3m	3m	3m	13.5m	33m
Quaternary	3m	3m	3m	3m	12m	30m

Table 3 Relative cost factors of CCD logic operators

Addition	Constant	Fixed overflow	Inhibit	Delay
2	1	4	6	2

TOLERANCE ANALYSIS AND RELATED MEASUREMENTS ON MVL-CCD'S *

H.G. Kerkhoff, J. de Groot and A.C. Brombacher

Twente University of Technology, Department of Electrical Engineering
Solid-State Electronics Group
7500 AE Enschede, The Netherlands

Abstract

*Prior to designing an MVL system on a single
silicon chip, it is necessary to perform a
feasibility study on the marginal reliability
aspects of the system to be implemented. In this
paper, the expected tolerances in basic CCD gate
configurations are investigated by a statistical
approach to the problem for which the Monte Carlo
analysis method has been chosen. The expected
tolerances were compared with actual data obtained
from measurements. As any MVL function can be
synthesized by means of these configurations, the
tolerance behaviour of a function can be investi-
gated by a general applicable analysis method
based on the Monte Carlo tolerance analysis
approach. The predecessor circuit was used to
illustrate the procedure followed.*

1. Introduction

In the design and implementation of multiple-
valued logic systems on a single silicon chip,
detailed knowledge about the influence of tolerances
of process and bias parameters on the overall
performance is of extreme importance. This results
from the fact that with single-chip implementations
it is usually not posssible to adjust components
afterwards in contrast to systems realized with
less complex IC's on a PC-board. A poor tolerance
and drift behaviour of a system can result in an
unacceptable low manufacturing yield and marginal
reliability. On the other hand, the system might
require such a large number of logic-level
regenerators in order to guarantee a good
tolerance behaviour, that the advantage of using
MVL devices in a specific application is seriously
diminished or can even turn out to be a disadvan-
tage.

The tolerance analysis and synthesis of MVL
circuits is evidently of vital importance. The
tolerance analysis is able to contribute to the
improvement of the tolerance behaviour of basic
transistor/gate configurations. In the tolerance
synthesis, tolerance requirements can be assigned
to process and bias parameters in order to obtain
a desired yield or marginal reliability.

However, few papers have been published
regarding this subject. This is probably due to
the fact that until now only a small number of
relatively simple MVL-IC's have actually been
realized. In addition, sufficient possibilities
were available on PC-boards to adjust voltages or
currents in order to compensate for tolerances of
parameters.

The influence of tolerances on the behaviour
of previously-designed MVL-I^2L circuits has been
investigated by Slob and Bos [1]. They performed
a so-called worst-case tolerance analysis on a
quaternary logic demultiplexer circuit and the
calculated tolerance behaviour turned out to be
discouraging for this circuit. These tolerance
problems stimulated the design of basic I^2L
transistor configurations which are less sensitive
to parameter tolerances [2].

*Worst-case calculations on MVL-I^2L circuits
(a threshold gate and a full adder) have also been
made by Dao [3] and Friedman [4] respectively.
Both concluded that the required tolerances for
their circuits could be met by I^2L technology.

A different approach, which also incorporated
the ambient temperature as a variable, has been
used by Russell in the investigation of the
tolerance behaviour of a ternary logic NMOS output
circuit [5]. He performed a deterministic worst-
case analysis of the output voltage of the circuit
by using a computer program for circuit analysis
(SLICM). It was then verified that the next
(digital) stage could be controlled by this output
voltage.

There is, however, one major drawback with
the above-mentioned approaches in the marginal
analysis of MVL-IC's: it is not possible to
incorperate probability-density functions (p.d.f.)
of parameters or correlation coefficients between
parameter values. However, the parameter value
p.d.f. and correlation coefficients are of special
importance in integrated circuits. Therefore the
previously obtained results do not seem to be very
realistic and/or accurate.

Except for a few publications concerning the
error rates in multiple-valued CCD memories [6, 7],
no papers have been published about the degradation

---

in performance of MVL-CCD's [8]. There are several reasons why the actual output of an MVL-CCD may deviate from the required performance:

- deviations resulting from aging (e.g. long-term drift of parameter values) or different environmental conditions (e.g. temperature and radiation effects).

- deviations resulting from a high value of the charge-transfer inefficiency ε [6] due to surface states and/or short transfer clock periods.

- deviations caused by tolerances in process and bias parameters, that primarily affect "yield" and in the second place drift-reliability.

The first problem will not be discussed in this paper as we shall assume normal environment conditions and neglect drift. In general, MVL surface-CCD's will operate at such clock frequencies, that the inefficiency ε per transfer will range between $1^o/oo$ and $1^o/o$. The deviations in the outputs of basic CCD gate configurations [8] as a result of tolerances in process and bias parameter values, are assumed at present to exceed the above-mentioned values of ε significantly. Therefore, the inefficiency ε will be neglected in the first instance. Hence, only the behaviour of MVL-CCD's with respect to tolerances in process and bias parameters will be treated. In order to avoid the previously-mentioned restrictions in the tolerance analysis methods as used for MVL-I^2L and NMOS circuits, a statistical, Monte Carlo tolerance analysis approach is chosen for applying to MVL-CCD's. This method is discussed in section 2 and subsequently applied to basic CCD gate configurations in section 3 in which also the simulated results are compared with actual measurements. In section 4, the Monte Carlo method is used to predict the circuit behaviour of the predecessor [8] when tolerances are assigned to the process and bias parameters. This simulated behaviour is compared with measurement results. Finally, conclusions and suggestions for further research are given.

2. Tolerance analysis methods

Tolerance analysis involves the investigation of the influence of tolerances of parameter or component values on the total performance of a circuit or system. Two different approaches have been suggested for tolerance analysis [9, 10]:

- the nonstatistical approach, which includes the worst-case method

- the statistical approach, of which the Monte Carlo method [11] is an example.

In the worst-case method, the extremes of a circuit performance are calculated with aid of the first order sensitivity coefficient for each component or parameter involved and the corresponding maximum deviation from its nominal value. The required calculations are relatively simple, but the results are often too conservative and uncertain, especially where nonlinear circuit behaviour (a change in sign of the partial derivatives of the function with respect to the parameters) and many parameters are concerned. Correlations between parameters are moreover not incorporated.

Although the sensitivities of some parameters of MVL-CCD circuits (e.g. the successor function) are known [12], they have never been applied in tolerance analysis; they were merely used to investigate which parameter required special attention in a design. Initial deterministic worst-case calculations on the multi-threshold CCD gate configurations show a serious degradation in charge levels.

The Monte Carlo tolerance simulation method is a computer-implemented procedure in which a system model is programmed and the outputs are computed for a sequence of sample functions from a stochastic input process [9, 10, 11]. The procedure is frequently used by IC manufacturers because it mathematically simulates the production process and its variations for many circuits. It is universally applicable to any network function, produces accurate results and is easy to use but usually requires a large amount of CPU time. A probability density function, together with the mean and standard deviation values can be specified for each parameter in the system model. Correlations between parameter values can also be incorporated. When the calculation of the output of the system is repeated for a large number of times and using the stochastic input variables, a histogram is obtained which is an approximation of the distribution of the output. The yield of the circuit can then easily be calculated from this distribution when the permitted tolerance of the output is specified. As the accuracy of the result is directly related to the number of calculation runs, the amount of required computer time can be very high for complex system models.

According to the formula which relates the number of computer runs to the accuracy [10], this number should be at least a few thousand in order to obtain a high accuracy. The large amount of computer time required is the only major drawback of the Monte Carlo tolerance analysis method.

The Monte Carlo procedure that we used to investigate the tolerance behaviour of MVL-CCD's was implemented in the following way:

- pseudorandom (uniformly distributed) numbers are generated by computer

- the numbers are transformed to pseudorandom numbers with a specified distribution, skewness, mean and standard deviation values which correspond to the actual circuit parameters [13]

- the mathematical relationship between input(s) and output(s) of the system is programmed in the computer in terms of process and bias parameters

153

- the probability density function of the
output is obtained by repeating the circuit
analysis of the system for a large number
of times, where the input variables are
obtained from the transformed pseudorandom
numbers.

The latter method has been applied to basic
CCD gate configurations and the predecessor func-
tion as it can provide much more realistic and
accurate results than the worst-case approach.
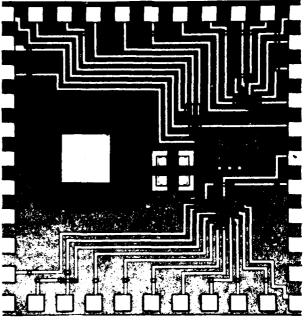
3. Tolerance analysis applied
to CCD gate configurations

Any MVL function can be implemented in CCD
technology with the aid of four basic CCD gate
configurations [8]. Hence, the tolerance behaviour
of the constant, multithreshold and inhibit confi-
gurations will contribute to the degeneration of
charge levels in a system and therefore these
configurations will be discussed in detail in this
section.

In order to perform a tolerance analysis, it
is necessary to know the distributions of process
and bias parameters as well as the equations that
describe the system behaviour. In table I, the
mean (μ) and standard deviation (σ) values of the
(p-type silicon) substrate concentration N_A, oxide
thickness t_{ox} and the flatband voltages V_{FB1} and
V_{FB2} [6] under the two different polysilicon gates
are listed. The distributions of the parameters
are assumed to be gaussian.

Table I: Mean and standard deviation values of the
process parameters used

	μ	σ
N_A (cm^{-3})	$1 \cdot 10^{15}$	$1 \cdot 10^{14}$
t_{ox} (nm)	100	3
V_{FB1} (V)	- 1.1	0.10
V_{FB2} (V)	- 1.9	0.15

It is emphasized that these parameters are specified
within a certain wafer. Especially the mean values
of the flatband voltages differ from wafer to wafer.
Usually, clusters of correlated doping and oxide
thickness values will occur on silicon wafers [14].
In the first instance, however, substrate dopings,
oxide thickness under and flatband voltages of
adjacent gates will be assumed to be uncorrelated.
The values of the flatband voltages were calculated
from static CV and threshold voltage measurements
performed on the test chip shown in fig. 1. The
chip also incorporates two basic CCD gate configu-
rations as well as read-in and read-out structures.
The chip is manufactured using the double poly-
silicon-level surface-CCD process and measures
2 by 2 mm.

200 μm

Fig. 1. photomicrograph of the test chip with the
incorporate two basic gate configurations
as well as read-in and read-out structures.

In the physical models of CCD gate configu-
rations, gate areas and power-supply voltages
appear. In the calculation of the effective gate
areas, effects like underetching and underdiffusion
have been taken into account. Although the design
of an accurate and stabilized clock generator and
dc power supply for MVL-CCD's in compatible NMOS
technology has already been accomplished (power
supply variation ratio of $\approx 10^{-3}$) [15], the
tolerances of voltages are at present set to
2.5% ($= 2\sigma$). The distributions are assumed to be
gaussian. Positive voltages are assumed to be
completely correlated with each other, while the
bulk voltage V_{bulk} is completely uncorrelated.

The CCD gate configuration models to be
discussed, are all based on the (one-dimensional)
calculation of the surface potential ψ_s under each
buried gate. This enables the introduction of
correlations and systematic shifts in parameters
(e.g. flatband voltages) at the cost of complex
and therefore long calculations. The relationship
between the surface potential ψ_s under a (surface)
CCD gate and the gate voltage V_G, bulk voltage
V_{bulk}, signal charge Q_n, N_A, t_{ox}, and V_{FB} is well
known [6] and given by:

$$\psi_s = V_G' + V_0 - \left[2 V_G' V_0 + V_0^2 \right]^{\frac{1}{2}} \quad (1a)$$

$$V_G' = V_G - V_{bulk} - V_{FB} + \frac{Q_n t_{ox}}{\epsilon_0 \epsilon_{ox}} \quad (1b)$$

$$V_0 = \frac{\epsilon_{si} q N_A (t_{ox})^2}{\epsilon_0 (\epsilon_{ox})^2} \quad (1c)$$

154

The hybrid drawing of the CCD gate configuration performing the generation of a *constant* in the charge domain (in this example $Q_o = < 3 >$) is shown in fig. 2.
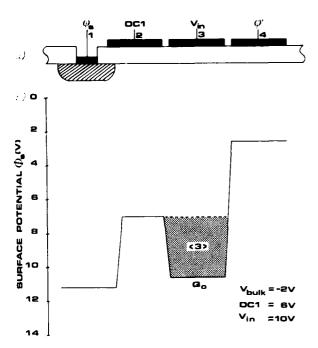

Fig. 2   Hybrid drawing of the gate configuration performing the generation of a constant in the charge domain

a) cross-section of the gate configuration
b) lateral surface-potential distribution after backflow of charge.

The bias voltages which are used in the model of the constant gate configuration have also been listed. The complete operation of this structure and the configurations to be discussed have already been explained in detail in another publication [16].

Besides the tolerance data regarding process and bias parameters, it is also required to provide the models that relate the input(s) and output(s) of the gate configurations in terms of these parameters. A very important design variable in digital CCD's is the charge-handling capacity C under a gate labelled g. For n-channel surface-CCD's, this capacity is given by [17]:

$$C = A_g \cdot \frac{\epsilon_o \epsilon_{ox}}{t_{oxg}} \left\{ (2 V_{og})^{\frac{1}{2}} \left[|\phi_{SF}|^{\frac{1}{2}} - |\phi_{SE}|^{\frac{1}{2}} \right] + \left[\phi_{SF} - \phi_{SE} \right] \right\} \quad (2)$$

where A_g, t_{oxg} and V_{og} denote respectively the effective area, oxide thickness and a material constant (eq. (1c)) of gate g. The surface potential ϕ_{SE} is the potential under gate g in the absence of charge and ϕ_{SF} the (lowest) surface

potential of the adjacent gates ($|\phi_{SF}| \leq |\phi_{SE}|$).

Gate g is equivalent to gate 3 in fig. 2 and the adjacent gate involved is gate 2. The output charge Q_o of the constant configuration is equivalent to C (eq. (2)). Hence Q_o is known in terms of process and bias parameters.

A Monte Carlo analysis was carried out for the constant < 3 > configuration using the equations (1) and (2) and the previously given process and bias parameter data. The result in the form of a histogram is shown in fig. 3.
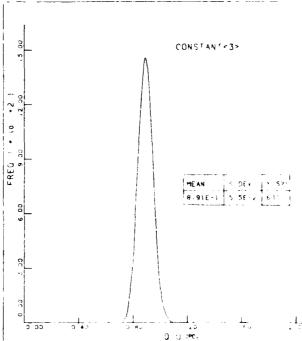

Fig. 3   Distribution of the output charge of a constant < 3 > configuration using the Monte Carlo simulation method. The mean, standard deviation and 3σ yield values are listed.

The output charge Q_o is plotted along the horizontal axis and the number of occurences along the vertical axis. The number of computer runs was 10,000 and the total CPU time 6 seconds.

The distribution of Q_o in fig. 3 is gaussian and the mean and standard deviation values are respectively 0.891 pC and 0.055 pC. It is also possible from the histogram in fig. 3, to calculate the yield of gate configurations which meet desired tolerance requirements. As an example, the yield was calculated for configurations of which the tolerance of Q_{out} is less or equal to 5% (Y = 63%).

The constant gate configuration has been incorporated in the test chip of fig. 1. Quasi-static current measurements at a clock frequency

of 100 kHz showed a mean and standard deviation value of Q_o of respectively 0.903 pC and 0.022 pC. The difference in calculated and measured mean values is probably caused by charge inefficiency and/or a small difference in the values of the actual flatband voltages and the ones used in the model. As correlations between process parameters of nearby gates have been neglected in first instance, the calculated standard deviation values are expected to be always somewhat larger than the measured values.

The hybrid drawing of the gate configuration performing the *multithreshold* operation in the charge domain (in this example $C_1 = <1>$ and $C_2 = <3>$) is shown in figure 4, together with the voltages involved. The circuit is discussed in detail in [6].
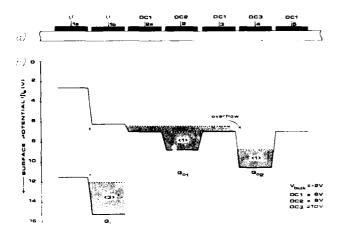
Fig. 4   Hybrid drawing of the gate configuration performing a multithreshold operation in the charge domain

a) cross-section of the gate configuration
b) surface potential in lateral direction during overflow of charge

The relationship between the input charge Q_i and the outputs Q_{om} (m = 1,2,..) of the generalized multithreshold gate configuration is given by [8, 16]:

$$Q_{om} = C_m \cdot Q_i^{\left[\sum_{k=1}^{m} C_k\right]} + (Q_i - \sum_{k=1}^{m} C_k + C_m) \cdot Q_i^{\left[\sum_{k=1}^{m} C_k - C_m, \sum_{k=1}^{m} C_k\right[} \quad (3)$$

It is obvious from eq. (3), that the charge-handling capacities C_k (k=1,2,..m) determine which specific multithreshold operation is performed. The expression for C_k can be obtained from eq. (1) and (2). The input and outputs are now again related to each other in terms of process and bias parameters. The Monte Carlo analysis was carried out for the multithreshold gate configuration (m = 1 and 2 and $C_1 = <1>$, $C_2 = <3>$) using table I and the equations (1), (2) and (3).

A representation for Q_{o1} and Q_{o2} similar to the one shown in fig. 3 is awkward as the outputs are dependent on the input charge Q_i. Therefore, mean and standard deviation values are abstracted from the distributions of Q_{o1} and Q_{o2} as function of Q_i. The results are shown in fig. 5a and 5b together with the actual measurement results.
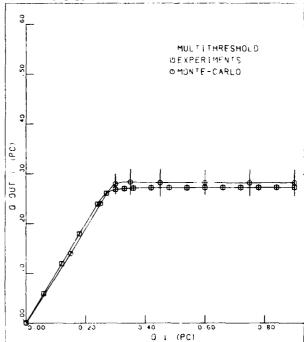

Fig. 5a   Monte Carlo simulation and actual measurement results of the output Q_{o1} of the multitreshold configuration ($C_1 = <1>$, $C_2 = <3>$) versus Q_{in}.

For the sake of clarity, the standard deviation values have only been plotted for a limited number of points. The number of computer runs was 12,000 and the total CPU time 76 seconds. Quasi-static current measurements at a clock frequency of 100 kHz were carried out on the multithreshold configuration which has been incorporated in the test chip of fig. 1. As shown in fig. 5a, the measured values of Q_{o1} are shifted slightly in the vertical direction with respect to the calculated values. This is probably caused by a mixture of effects, such as slightly different actual flat band voltages, dark currents and the influence of charge inefficiency. The shift of the measured values of Q_{o2} in the horizontal axis with respect to the calculated values is probably due to the small (measured) charge-handling capacity of C_1. As fig. 5 shows, the measured mean and standard deviation values fall within the values of the Monte Carlo simulations. For the difference between the calculated and measured standard deviation values, the same comments hold as given for the constant generator.
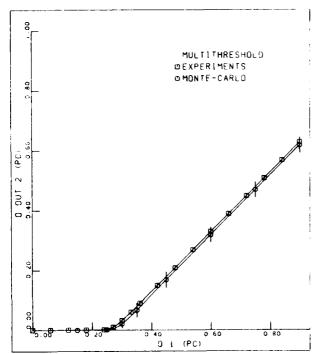
156

Fig. 5b   Monte Carlo simulation and actual
           measurement results of the output
           Q_{o2} versus Q_{in}.

The gate configuration performing the
inhibit operation in the charge domain consists of
a sensing and a control part. The hybrid drawings
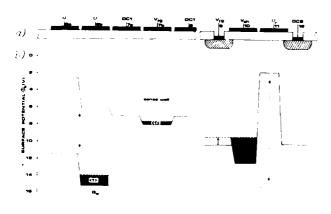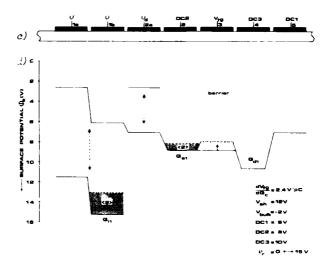of these parts are shown in fig. 6a, b and 6c, d
respectively.

Fig. 6   Hybrid drawing of the sensing and control
          part of the gate configuration performing
          the inhibit operation in the charge domain

          a) cross-section of the sensing part
          b) corresponding surface potential in
             lateral direction

Fig. 6   c) cross-section of the control part
          d) corresponding surface potential in
             lateral direction.

The values of the dc voltages and reset pulse
involved in the tolerance analysis as well as the
floating gate sensitivity $\frac{d\,V_{fg}}{d\,Q_c}$ [12] are listed
in fig. 6. A detailed explanation of this gate
configuration is given in reference [16]. A close
look at fig. 6c, d and fig. 4 shows that the
control structure is almost identical to the
previously-discussed multithreshold gate configura-
ration. The difference lies in the status of
(barrier) gate 3, which is electrically connected
with gates 7b and diffused region 9 (fig. 6a, b,
c, d). In contrast to the multithreshold gate
configuration, in which a fixed bias voltage is
applied to gate 3, the potential V_{fg} depends on the
value of Q_c. Hence, the relationship between V_{fg}
(or the corresponding surface potential under
gate 3) and Q_c has to be found in terms of process
and bias parameters. This is, however, a complex
problem.

When the (shielded) reset MOS transistor
(9, 10, 11, 12) is switched off, two shifts of
V_{fg} from the reset potential DC2 (applied to
drain 12 of the MOST) will occur:

- a shift (V_{inv}) in V_{fg} as result of inversion
  charge of the reset MOST which is independent
  of the control charge Q_c
- a shift $\left[\frac{d\,V_{fg}}{d\,Q_c}\,.\,Q_c\right]$ in V_{fg} as result of
  the control charge Q_c.

Hence, the floating-gate voltage V_{fg} can be
written as [18]:

$$V_{fg} = DC_2 + V_{inv} + \frac{d\,V_{fg}}{d\,Q_c}\,.\,Q_c \qquad (4)$$

In order to avoid the influence of tolerances on the blocking behaviour of the inhibit configuration, the floating-gate swing $\left[\dfrac{d\ V_{fg}}{d\ Q_c}\cdot Q_c\right]$ is usually larger than strictly required for proper operation.

The voltage swing V_{inv} is a function of the dimensions of the reset MOS transistor, V_{sh}, reset pulse ϕ_r, its fall-off time and the total floating-gate capacitance C_{fg} with respect to the substrate [18]. The latter consists of the total oxide and depletion capacitances of the sensing and barrier gates and a (parasitic) rest capacitance C_r.

The related floating-gate sensitivity $\dfrac{d\ V_{fg}}{d\ Q_c}$ for a simple floating gate is given by [6, 12, 17, 18]:

$$\frac{d\ V_{fg}}{d\ Q_c} = \frac{C_{oxs}}{C_r\cdot C_{oxs}+A_s\cdot C_{ds}\cdot C_{oxs}+C_r\cdot C_{ds}} \tag{5}$$

where C_{oxs}, C_{ds} and A_s denote respectively the oxide and depletion capacitance and the area of the sensing gate.

The relationship between Q_i, Q_s, Q_d and the well capacity C_1 under gate 2 (fig. 6c,d) in an inhibit configuration is (ideally) given by:

$$Q_s = \begin{cases} Q_i & \text{if} & C_1 \geq Q_i \\ 0 & & C_1 = 0 \end{cases} \tag{6a}$$

$$Q_d = \begin{cases} 0 & \text{if} & C_1 \geq Q_i \\ Q_i & & C_1 = 0 \end{cases} \tag{6b}$$

The charge-handling capacity C_1 is dependent on Q_c, through eq. (4) of the potential V_{fg} and can be calculated by using eq. (1) and (2). Hence, Q_s and Q_d as function of Q_i and Q_c are known in terms of process and bias parameters. For the inhibit configuration in which a floating diffusion structure [16] is incorporated, the same procedure can be followed.

The Monte Carlo tolerance analysis was applied to the inhibit configuration of fig. 6 and the results are shown in fig. 7. The outputs Q_s and Q_d are plotted as function of Q_i with Q_c as parameter in fig. 7a and 7b respectively. The number of computer runs was 20,000 and the CPU time 3 minutes and 30 seconds. A stand-alone, standardized inhibit configuration as shown in fig. 6 has not been incorporated yet on a test chip and therefore no direct measurement results are available.

The switching behaviour of the inhibit configuration as function of Q_c can be clearly recognized in fig. 7. The control charge Q_c is increased from 0 to 0.45 pC in discrete steps of 0.15 pC. When the value of Q_c lies between 0 and 0.3 pC, the inhibit configuration operates as a *programmable multi-threshold* [8]. The standard deviation values are high in this operating range, as can be seen in fig. 7. It illustrates why this operation has a high cost factor [8] and is usually avoided in MVL-CCD's.

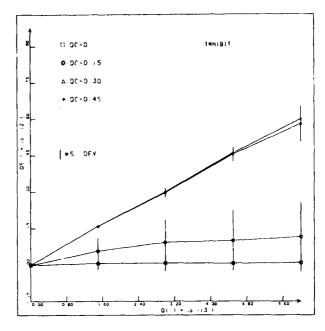Fig. 7   a) Monte Carlo simulation of the output charge Q_s of the inhibit gate configuration as function of Q_i with Q_c as parameter.
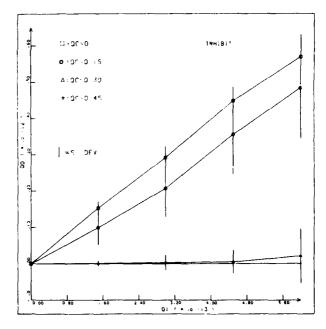

Fig. 7   b) Monte Carlo simulation of the output charge Q_d as function of Q_i with Q_c as parameter.

4. The tolerance analysis of MVL-CCD functions

In the past, several methods have been developed to decompose any MVL function into basic CCD gate configurations [8]. A simple example of a decomposition scheme is seen in fig. 8.
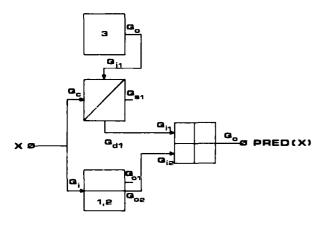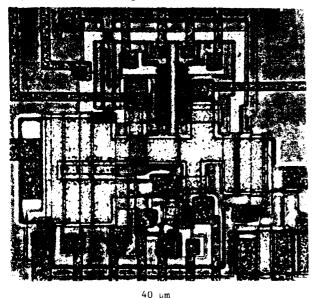

Fig. 8   Decomposition scheme of the predecessor function in quaternary logic.

It is the scheme of the circuit which performs the predecessor function in the charge domain in radix four. A photomicrograph of the actual circuit layout is shown in fig. 9.

40 µm

Fig. 9   Photomicrograph of the predecessor circuit including read-in and read-out circuits.

The occupied chip area is less than 0.01 mm² using 10 µm minimum dimensions.

The (Pascal based) computer programs developed for the (Monte Carlo) tolerance analysis of basic CCD gate configurations have been combined and extended in order to quickly simulate the tolerance behaviour of multiple-valued logic CCD functions starting from the decomposition schemes. The distributions of charges at each point of the decomposition scheme can be recalled, which is important in order to determine the optimum location of logic-level regenerators. This (interactive) computer program was applied to the predecessor circuit and the results are seen in fig. 10.
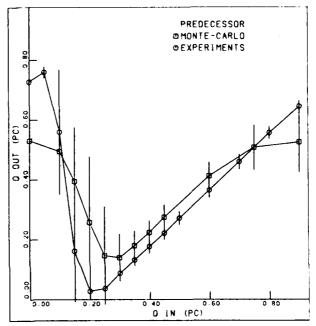

Fig. 10   Monte Carlo simulation and actual measurement results of the output charge Q_o versus the input charge Q_i of the predecessor circuit.

The input charge Q_i is plotted along the horizontal axis and the mean and standard deviation values of the output charge Q_o along the vertical axis, together with the measured results. The number of computer runs was 12,000 and the required CPU time 2 minutes.

The input and output signals of a complete CCD operate in the voltage domain. A photograph of the input and output voltages of the predecessor circuit, working at a clock frequency of 50 kHz, is seen in fig. 11. Because equal flat band voltages were assumed for the first and second poly silicon gates during the design of the circuit, the gate voltage V_{in} (fig. 2) of the constant < 3 > gate configuration (fig. 8) has been increased in fig. 11 to compensate for the resulting effects. The total delay between input and output voltage is two clock-pulse periods. The influence of surface
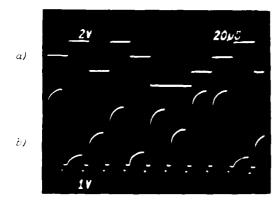
159

*Fig. 11   Input and output signals of the
quaternary logic predecessor circuit
(f_c = 50 kHz)*

*a) quaternary logic input signal
b) output signal of the predecessor
(delay 2 clock pulse periods)*

states on the behaviour of the predecessor is
clearly seen in fig. 11b. In order to investigate
the relationship between the charges in fig. 10
and the input and output voltages of the (non-com-
pensated) predecessor circuit, the behaviour of
the read-in and read-out structures must be
accurately known. These structures have been
incorporated for this purpose in the test chip as
shown in fig. 1. The measured mean and standard
deviation values of the sensitivities of the
read-in circuit are 0.203 pC/V and 0.004 pC/V and
of the read-out circuit 3.99 V/pC and 0.22 V/pC
respectively.

The curve representing the behaviour of the
output charge of the predecessor circuit as
function of the input charge as shown in fig. 10,
can be divided into two regions:

- in the first region (Q_i < 0.3 pC), the inhibit
  and constant < 3 > gate configuration dominate
  the logic behaviour. As a result of the
  difference in flat band voltages, the charge-
  handling capacity of the constant < 3 > confi-
  guration is larger than calculated. The
  standard deviations in the switching region
  are large as can be seen in fig. 10. This
  results from the calculation methods as used
  in the Monte Carlo analysis, in which the
  average charge levels are calculated in
  inhibit configurations which are either switched
  on or off.

- in the second region (Q_i > 0.3 pC), the multi-
  threshold configuration (C_1 = < 1 > and C_2 = < 2 >
  (fig. 8) dominates the logic behaviour. The
  differences between the calculated and measured
  values of the charges of the multithreshold
  configuration are probably caused by
  deviations in flat band voltages and charge
  transfer inefficiencies.

In a similar way as discussed for the
predecessor circuit, the tolerance behaviour and
associated yield at each point of any MVL-CCD
circuit or system can be investigated by using
this extended Monte Carlo tolerance analysis
computer program.

5. Conclusions

The tolerance behaviour of basic CCD gate
configurations has been investigated by means of
the Monte Carlo tolerance analysis method. In can
be used to improve the behaviour of these configu-
rations. The calculated results have been compared
with measurement data. Deviations in flat band
voltages under first and second poly silicon
gates and the presence of dark currents and charge-
transfer inefficiencies are suspected to contribute
to the differences between calculated and measured
values.

A computer program was developed to
calculate the tolerance behaviour of MVL-CCD
functions and systems from the decomposition
schemes with little effort. The results can be
used to locate the positions of logic-level
regenerators in an MVL system. It was found that
differences in flat band voltages play an important
role in the logic behaviour of MVL-CCD's. The
influence of dark currents, surface states and
speed limiting efforts on the overall logic
behaviour of MVL-CCD's cannot be neglected and
therefore the system models should be extended
in the near future to incorporate their effects.

References

[1] A. Slob, G.A.A. Bos, "Four-valued logic",
     Philips Research Laboratories, technical
     note No. 235/77, unpublished.

[2] C.A. Johnson, J.R. Armstrong, "Improved I^2L
     for multivalued logic", in Proc. of the 11th
     International Symposium on Multiple-Valued
     Logic, Oklahoma City (OK), May 1981, pp.
     200-204.

[3] T.T. Dao, "Threshold I^2L and its applications
     to binary symmetric functions and multivalued
     logic", IEEE Journal of Solid-State Circuits,
     Vol. SC-12, No. 5, Oct. 1977, pp. 463-472.

[4] N. Friedman, C.A.T. Salama, F.E. Holmes and
     P.M. Thompson, "Realization of a multivalued
     integrated injection logic (M I^2L) full
     adder", IEEE Journal of Solid-State Circuits,
     Vol. SC-12, No. 5, Oct. 1977, pp. 532-534.

[5] L.K. Russell, "Multilevel NMOS circuits",
COMPCON-Spring, San Francisco, USA, Feb.
1981, unpublished.

[6] C.H. Séquin, M.F. Tompsett, "Charge transfer
devices", Academic Press, New York, 1975,
pp. 238-243.

[7] H.S. Abdel-Aty Zohdy, S.G. Chamberlain and
L.A.K. Watt, "Limitations of multilevel
storage in charge-coupled devices",
IEEE Transactions on Electron Devices,
Vol. ED-27, No.9, Sept. 1980, pp. 1733-1743.

[8] H.G. Kerkhoff, H.A.J. Robroek, "The logic
design of multiple-valued logic functions
using charge-coupled devices", in Proc. of
the 12th International Symposium on
Multiple-Valued Logic, Paris, France, May
1982, pp. 35-44.

[9] S.T. Li, "Computer-aided tolerance assign-
ment and analysis", Ph.D. dissertation,
Georgia Institute of Technology, May 1974.

[10] H.P. Herrmann, "Ausbeuteberechnung und
Optimierung von integrierten Schaltungen",
Ph.D. dissertation, Universität (TH)
Karlsruhe, May 1980.

[11] J.M. Hammersley, D.C. Handscomb, "Monte
Carlo methods", John Wiley & Sons Inc,
New York, 1964.

[12] H.G. Kerkhoff, M.L. Tervoert and H.A.C.
Tilmans, "Design considerations and
measurement results of multiple-valued
logic CCD's", in Proc. 11th International
Symposium on Multiple-Valued Logic,
Oklahoma City, OK, May 1981, pp. 205-211.

[13] J. Nievergeldt, J. Farrar, and E.M. Reingold,
"Computer approaches to mathematical
problems", Prentice Hall, 1974, pp. 151-152.

[14] D.S. Perloff, F.E. Wahl and J.D. Reimer,
"Contour maps reveal non-uniformity in
semiconductor processing", Solid-State
Technology, Feb. 1977, pp. 31-36.

[15] F.J. List, "Design and simulation of a
voltage stabilization and clamp circuit
in single-channel MOS technology", (in Dutch),
Twente University of Technology, report
nr. 1222.3301, Enschede, May 1982.

[16] H.G. Kerkhoff, "Theory and design of
multiple-valued logic CCD's", in "Computer
science and multiple-valued logic: theory
and applications"", Chapter 16, edited by
D.C. Rine, North-Holland Pub., to be
published.

[17] R.J. Handy, "Charge sensing amplifier",
Ph.D. thesis, University of California,
Los Angeles, 1977.

[18] R.A.J. Gal, "Simulations and measurements
on floating gate structures", Twente
University of Technology, report nr.
1217.3346, Enschede, Feb. 1980.

TABULAR METHODS FOR THE DESIGN OF CCD MULTIPLE-VALUED CIRCUITS

Joo-kang Lee and Jon T. Butler

Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60201

ABSTRACT

A tabular method for the design of multiple-valued CCD circuits is introduced which produces less expensive realizations with a significantly smaller table than a method proposed by Robroek [4] and Kerkhoff and Robroek [3]. In addition, a universal cost table method is shown which produces even less expensive realizations, but which has a considerably longer table. Further, a flexible CCD circuit is shown which produces any unary function by a simple adjustment of voltage levels.

I. INTRODUCTION

In multiple-valued CCD circuits, logic levels are represented by quantities of charge. Operations, such as charge addition and charge overflow, can be combined to form combinational logic devices. Because preservation of a logic level does not depend on a flow of charge, as in T^2L and I^2L technologies, very little power is consumed. As a result, CCD is well suited to high density VLSI implementation. Problems such as low speed and logic level deterioration do exist and remain to be solved. In spite of this, multiple-valued CCD technology is an area of considerable interest.

In a pioneering paper, Kerkhoff and Tervoert [1] described the implementation of a set of 4-valued CCD circuits which comprise a logically complete set of gates. The sensitivity of such gates to manufacturing imperfections and power supply fluctuations, was described in Kerkhoff, Tervoert, and Tilmans [2]. CCD technology has now matured to the point here logic design techniques have been given careful study. In Kerkhoff and Robroek [3], synthesis techniques for other technologies were adapted to CCD. Similarly, Ishizuka [5] has considered the synthesis of unary CCD circuits using multithreshold design techniques. A synthesis technique expressly for one-input 4-valued CCD circuits was presented in [3] and, in greater detail, in Robroek [4]. It is based on the decomposition of a given function into subfunctions, which, in turn, are realized from a set of four basic logic operations. A cost is associated with each basic operation that is based on fabrication complexity. For example, a basic operation which occupies a large area tends to have a high cost. The synthesis technique produces realizations with small cost. Unlike previous algebraic techniques [6,7,8], this technique is tabular.

In this paper, an improved version of the tabular technique [3,4] is presented, which produces lower cost realizations. Also, a universal cost table approach is described which produces even lower cost realizations. These techniques are compared on the basis of the average cost of the realizations produced and on algorithm complexity. Additionally, a programmable circuit is introduced and compared with the above techniques. Although the discussion will concentrate on one-input functions, the synthesis can be extended to n-input functions for n > 1 in a straightforward way.

II. BACKGROUND

Fig. 1 shows four basic operations which are the basis of the two synthesis techniques proposed here. These operations are the set of five described in [1,2] less the linear programmable overflow. This has been omitted, as in [3], because of its high relative cost. The set shown in Fig. 1 is functionally complete [1]; any 4-valued function can be realized from this set. The notation here used here is identical to that used in [3]:

$$_0\alpha] = \begin{cases} 1 & \text{if } 0 < \alpha \\ 0 & \text{otherwise} \end{cases}$$

$$_0]\alpha = \begin{cases} 1 & \text{if } 0 > \alpha \\ 0 & \text{otherwise} \end{cases}$$

$$_0[\alpha, \beta[= \begin{cases} 1 & \text{if } \alpha < 0 < \beta \\ 0 & \text{otherwise} \end{cases}$$

The synthesis technique proposed in this paper is presented in more detail in Lee [9]. It consists of the successive decomposition of a given function f(x) into restfunctions which in turn may be further decomposed. The process continues until a restfunction occurs which appears in the table. For example, Fig. 2 shows the binary tree of an example decomposition. Here, f(x) is divided into restfunctions $R_1(x)$ and $R_2(x)$, neither of which are in the table. However, $R_2(x)$ can be divided into two functions, $S_1(x)$ and $S_2(x)$, both of which are in the table. Thus, the tree terminates at $S_1(x)$ and $S_2(x)$. The function at a parent node is the sum of functions at the two daughter nodes (using the addition operation in Fig. 1). Therefore, summing the functions $S_1(x)$, $S_2(x)$, $S_3(x)$, $S_4(x)$, and $S_5(x)$ in Fig. 2, produces f(x).

162

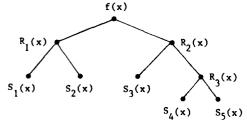| OPERATION & DECOMPOSITION LEVEL	RELATIVE COST FACTOR		MATHEMATICAL LEVEL
	CONDITION	COST	
ADDITION	n = 2	2	$$Q_o = \sum_{m=1}^{n} Q_{1m}$$ $$n = 2, 3, \ldots$$
CONSTANT	L = 1, 2, 3	1	$$Q_o = L$$ $$L = 1, 2, \ldots$$
FIXED OVERFLOW	r = 3 $C_1 = C_2 = C_3 = 1$	4	$$Q_{om} = C_m \cdot (\sum_{k=1}^{m} C_k)^1 + (Q_1 - \sum_{k=1}^{m-1} C_k) \cdot Q_1^{[\sum_{k=1}^{m-1} C_k \cdot \sum_{k=1}^{m} C_k [}$$ $$m = 1, 2, \ldots, r \qquad r = 1, 2, \ldots \qquad C_k = 0, 1, \ldots$$
INHIBIT	n = 1 ; $Q_1 < 2$	6	$$Q_{sm} = Q_{1m} \cdot Q_c^{10}$$ $$Q_{dm} = Q_{1m} \cdot Q_c^{01]}$$
	n = 1 ; $Q_1 < 3$	23	$$m = 1, 2, \ldots, n \qquad n = 1, 2, \ldots$$

Figure 1. Four Basic Operations in 4-Valued CCD.

Figure 2. Example of the Decomposition of a Function f(x) Into Five Cost Table Functions.

It is convenient to represent a one-variable 4-valued function as a 4-tuple $\langle x_0, x_1, x_2, x_3 \rangle$, where 0, 1, 2, and 3 map to x_0, x_1, x_2, and x_3, respectively. There are three types of one-variable functions.

Definition: A function $\langle x_0, x_1, x_2, x_3 \rangle$ is an **m-increasing** (**m-decreasing**) function if and only if $x_0 < x_1 < x_2 < x_3$ ($x_0 > x_1 > x_2 > x_3$); otherwise it is an **m-mixed** function, where $m = \text{Max}\{x_0, x_1, x_2, x_3\} - \text{Min}\{x_0, x_1, x_2, x_3\}$.

For example, $\langle 0,0,2,3 \rangle$, $\langle 3,2,0,0 \rangle$, and $\langle 0,2,3,0 \rangle$ represent 3-increasing, 3-decreasing, and 3-mixed functions, respectively. Note that the four constant functions are each 0-increasing and 0-decreasing.

Definition: The **transition count** (TC) of a mixed function is the number of times the trend in output logic values changes from decreasing to increasing or vice versa plus 1 if the function is initially decreasing.

In general, the cost of an increasing function is less than that of a decreasing function, which in turn, is less than that of a mixed function. Increasing functions require fixed overflows only, plus perhaps an adder. Decreasing functions require, in addition, inhibit circuits. Within the set of mixed functions, those with a higher TC generally, have a higher cost because of the additional inhibits needed for each transition. This can be seen in the preview of the synthesis technique shown in Table I on the next page.

The number of increasing, decreasing, and mixed m-valued unary functions can be calculated as follows. Consider first the enumeration of increasing functions. Rather than counting such functions directly, we prefer to establish a one-to-one correspondence with sequences of values whose number is determined in a straightforward way.

An increasing function f has the form $f = \langle f_0, f_1, \ldots, f_{m-1} \rangle$, where $f_1 < f_{i+1}$. Appending to f the sequence 0, 1, ..., m-1 and rearranging in increasing order, produces an ordered sequence of length 2m, where every value, 0, 1, ... ,m-1, occurs at least once. Let F_m denote the set of

163

Example 1:

Function	TC	Type	Cost	Decomposition	Composition (adders are not shown)
$\langle 0,0,2,3 \rangle$	0	increasing	10	$\langle 0,0,1,1 \rangle$, $\langle 0,0,1,2 \rangle$	2 fixed overflows
$\langle 3,2,0,0 \rangle$	1	decreasing	20	$\langle 2,2,0,0 \rangle$, $\langle 1,0,0,0 \rangle$	1 fixed overflow & 2 inhibits
$\langle 0,2,3,0 \rangle$	1	mixed	22	$\langle 0,1,1,0 \rangle$, $\langle 0,1,2,0 \rangle$	2 fixed overflows & 2 inhibits
$\langle 0,2,0,3 \rangle$	2	mixed	29	$\langle 0,0,0,1 \rangle$, $\langle 0,2,0,2 \rangle$	3 fixed overflows & 2 inhibits
$\langle 2,0,3,0 \rangle$	3	mixed	38	$\langle 2,0,0,0 \rangle$, $\langle 0,0,2,0 \rangle$, $\langle 0,0,1,0 \rangle$	2 fixed overflows & 4 inhibit

Table I. Results of the Synthesis Technique on Example Functions ☐

n-valued increasing functions, and let S_m denote the set of ordered sequences described above. then, $F_m \subset S_m$. conversely, for a given sequence S_m, eliminating one copy of each i, for $0 < i < m-1$, yields a distinct increasing function in F_m. Thus, $S_m \subset F_m$, and we have $F_m = S_m$.

An increasing sequence in S_m can be specified as a set of 2m positions separated into m groups by m-1 dividers. The dividers mark the transition from i to i+1, for $0 < i < m-2$. Since there are 2m - 1 spaces between the 2m positions, $S_m = C(2m-1, m-1)$, where $C(n,r)$ is the number of combinations of n things taken r at a time. Thus, the number of increasing functions N_{inc} is given as

$$N_{inc} = C(2m-1, m-1). \qquad (1)$$

Applying Sterling's approximation yields,

$$N_{inc} \simeq 4^m / 2(\pi m)^{1/2}, \qquad (2)$$

when m is large. (2) is also reasonably accurate for m as small as 4.

The set of decreasing functions are built up in an analagous way, and we have

$$N_{dec} = N_{inc}.$$

Since the total number of unary functions is m^m, the fraction which are either increasing or decreasing,

$$(4/m)^m / 2(\pi m)^{1/2},$$

becomes vanishingly small as m approaches infinity. However, for m = 4, the fraction of functions which are either increasing or decreasing is a substantial percentage, 25%.

Consider now the synthesis technique. Three principles drive the decomposition. At each step, where a function f is divided into two restfunction r_1 and r_2;

1. the TC of both r_1 and r_2 should be less than (or at most equal to) that of f,

2. if f contains a 3 (or larger value), the corresponding values in r_1 and r_2 should be 2 or 1

(smaller), unless a cost table function would otherwise be obtained. For example, under this principle, 6-valued function $\langle 1,0,3,2,5,4 \rangle$ would be divided into two functions containing 0's, 1's, 2's, and 3's, while $\langle 4,5,4,4,5,4 \rangle$ would be divided into a function containing 0's and 1's and $\langle 4,4,4,4,4,4 \rangle$, since the latter is a low-cost table function, and

3. the functions r_1 and r_2 should be chosen to minimize the total number of component functions. Dividing the logic values equally among r_1 and r_2 tends to do this.

III. THE COST TABLE

The basis for the technique reported in Kerkhoff and Robroek [3] and Robroek [4] is a table of 45 four-valued one-variable functions. Associated with each is a cost which is the sum total of the costs of the basic operations that make up the function. Such a table is called a **cost table**. In this section, we introduce a cost table of only 24 functions and show that the functions produced have a lower cost, on the average, than that of [3,4].

Common to both the cost table of [4] and the one presented here are the four unit functions, $\langle 1,0,0,0 \rangle$, $\langle 0,1,0,0 \rangle$, $\langle 0,0,1,0 \rangle$, and $\langle 0,0,0,1 \rangle$. From these, all 256 one-variable functions can be synthesized. Thus, a cost table of size four is sufficient for the realization of all functions. However, target functions synthesized from this table will be quite costly since many unit functions will be required. The addition of functions to the table which are realized more efficiently by other combinations of basic operations results in more efficient realizations of still more functions. Ultimately, one can have a table of 256 functions, in which case the synthesis of a one-variable function f consists of just looking for f in the table. The latter is a **universal cost table**. This larger table produces the lowest cost realizations of unary functions, but it is considerably longer than either of the other two tables described here.

For ease of representation, we adopt the notation of [4] for the four basic operations:

164

C(L)   ; Constant L generation

$A(Q_1, Q_2, Q_3)$; Addition of three input charges Q_1, Q_2, and Q_3.

$I(Q_1, Q_c, Q_o)$; Inhibit where, Q_1, Q_c, and Q_o are the input, control, and output charge respectively.

$F(Q_x, C_1, C_2, C_3, Q_1, Q_2, Q_3)$; Fixed overflow where Q_x is the input charge, C_1, C_2, and C_3 are optional well capacities, and Q_1, Q_2, and Q_3 are optional outputs.

      Shown below is the cost table of 24 functions. For each entry, both the function cost and its realization are shown.

No. Function Cost Realization

No.	Function	Cost	Realization
1	1 1 1 1	1	$C(1)$
2	2 2 2 2	1	$C(2)$
3	3 3 3 3	1	$C(3)$
4	1 2 3 3	3	$A(1, Q_x)$
5	2 3 3 3	3	$A(2, Q_x)$
6	0 0 0 1	4	$F(Q_x, 1, 1, 1, Q_3)$
7	0 0 1 1	4	$F(Q_x, 1, 1, Q_2)$
8	0 1 1 1	4	$F(Q_x, 1, Q_1)$
9	0 0 1 2	4	$F(Q_x, 1, 2, Q_2)$
10	0 1 2 2	4	$F(Q_x, 2, Q_1)$
11	0 1 1 2	6	$F(Q_x, 1, 1, 1, Q_1, Q_3)$; $A(Q_1, Q_3)$
12	0 1 3 3	6	$F(Q_x, 1, 2, Q_2)$; $A(Q_2, Q_x)$
13	0 2 3 3	6	$F(Q_x, 1, Q_1)$; $A(Q_1, Q_x)$
14	1 0 0 0	7	$I(1, Q_x, Q_d)$
15	2 0 0 0	7	$I(2, Q_x, Q_d)$
16	0 2 2 2	7	$I(2, Q_x, Q_s)$
17	0 0 1 1	10	$F(Q_x, 1, 1, 1, Q_2, Q_3)$; $I(Q_2, Q_3, Q_d)$
18	0 1 0 0	10	$F(Q_x, 1, 1, Q_1, Q_2)$; $I(Q_1, Q_2, Q_d)$
19	0 1 1 0	10	$F(Q_x, 1, 1, 1, 0, Q_3)$; $I(Q_1, Q_3, Q_d)$
20	0 1 2 0	10	$F(Q_x, 2, 1, Q_1, Q_2,)$; $I(Q_1, Q_2, Q_d)$
21	1 1 0 0	11	$F(Q_x, 1, 1, Q_2)$; $I(1, Q_2, Q_d)$
22	1 1 1 0	11	$F(Q_x, 1, 1, 1, Q_3)$; $I(1, Q_3, Q_d)$
23	2 2 0 0	11	$F(Q_x, 1, 1, Q_2)$; $I(2, Q_2, Q_d)$
24	2 2 2 0	11	$F(Q_x, 1, 1, 1, Q_3)$; $I(2, Q_3, Q_d)$

Table II. Cost Table Used in the Synthesis Techniques

IV. SYNTHESIS OF UNARY FUNCTIONS USING COST TABLE

      The synthesis technique is described in ALGOL in Table III. Although it appears to be complicated, it is not.

      The constant functions $\langle 1,1,1,1 \rangle$, $\langle 2,2,2,2 \rangle$, and $\langle 3,3,3,3 \rangle$ are synthesized trivially; each is in the cost table.

      An increasing function f which is not a constant function can be split into a pair of functions C_i and C_j whose sum is f, where C_i and C_j are both in the cost table or one of C_i and C_j is in the table and the other is a constant times a function in the cost table. In either case, a

search produces the minimal cost realizations.

      A decreasing function which is not a constant function is handled in the same way except for three functions $\langle 2,1,0,0 \rangle$, $\langle 2,1,1,0 \rangle$, and $\langle 3,2,1,0 \rangle$, whose minimum realization involves $\langle 2,0,0,0 \rangle$.

      For a mixed function f, the procedure is different, depending on whether f has 0's and where they are located.

procedure DECOMPOSITION (f)

```
// f(x) is a target function to be synthesized.  //
// M(x) is a multiple function.                  //
// R_i(x) is a restfunction, where i = 1,2,...    //
// T is the cost table.                          //
// C_i(x) is a function in the cost table.        //
// I is the set of all increasing function.      //
// D is the set of all decreasing function.      //
```

1   **if** $f(x) \in T$ **then return**
2   **if** $f(x)/2 \in T$ **then call** MULTIPLE (f); **return**
3   **if** $f(x) \in I$ **then** /f(x) is increasing function/
4   **if** $\exists \{C_i(x), C_j(x) \in I\} [C_i(x) + C_j(x) = f(x)]$
          /if there exist $C_i(x)$ & $C_j(x)$ which are in I, such that they form f(x)/
5   **then** choose lowest cost pair; **return**
6   **else** split f(x) into $C_i(x)$ & M(x) such that $C_i(x) \in I$ and $M(x)/2 \in T$;
7   **call** MULTIPLE (M); **return**

8   **if** $f(x) \in D$ **then** /f(x) is decreasing function/
9   **case**
10   :$f(x) = \langle 2,1,0,0 \rangle, \langle 2,1,1,0 \rangle,$ or $\langle 3,2,1,0 \rangle$: split f(x) into $\langle 2,0,0,0 \rangle$ & R(x)
11   **call** DECOMPOSITION (R); **return**
12   :$x_3 \neq 0$: split f(x) into $\langle x_3, x_3, x_3, x_3 \rangle$ & R(x);
13   **call** DECOMPOSITION (R); **return**
14   :**else**: split f(x) into $C_i(x)$ & $C_j(x)$ such
15   that $C_i(x), C_j(x) \in D$; **return**
          /there are six decreasing functions to be candidates in T /
16   **end**

      / if f(x) is mixed function, then there are several cases to be considered/
17   **case**
18   :$x_0 = x_1 = 0$ or $x_2 = x_3 = 0$:
19   **if** $\exists \{C_i(x), C_j(x)\} [C_i(x) + C_j(x) = f(x)]$
20   **then return**
21   **else** split f(x) into $C_i(x)$ & the largest value of M(x) $[M(x)/2 \in T]$;
          call MULTIPLE (M); **return**

22   :$x_1 = 0$ or $x_2 = 0$:
23   **if** $x_0 < 1, x_1 > 2, x_2 = 0,$ and $x_3 > 2$
24   **then** split f(x) into $\langle 0,2,0,2 \rangle$ & R(x);
25   **call** MULTIPLE ($\langle 0,2,0,2 \rangle$);
26   **call** DECOMPOSITION (R); **return**
27   **else** split f(x) into $\langle x_0, x_1, 0, 0 \rangle$ & $\langle 0, 0, x_2, x_3 \rangle$;
28   **call** DECOMPOSITION ($\langle x_0, x_1, 0, 0 \rangle$);
29   **call** DECOMPOSITION ($\langle 0, 0, x_2, x_3 \rangle$);
          return

```
30      :x₀ = 0 or x₃ = 0:
31        case
32          :f(x) = <0,2,1,3>:
                 split f(x) into <0,1,1,2> & <0,1,0,1>;
33              call DECOMPOSITION (<0,1,0,1>); return
34          :f(x) = <2,1,3,0> or <3,1,2,0>:
35              split f(x) into <2,0,0,0> & R(x);
36              call DECOMPOSITION (R); return
37          :}(C_i(x), C_j(x)) [C_i(x) + C_j(x) = f(x)]:
38              choose lowest cost pair; return
39          :}(C_i(x), M(x)) [M(x)/2 ε T and C_i(x) +
                 M(x) = f(x)]:
40              call MULTIPLE (M); return
41          :else:  split f(x) into <x_1,x_1,x_1,0> or
                 <0,x_1,x_1,x_1> & R(x) such that
                 Min[x_0, x_1, x_2, x_3] = x_1 ≠ 0;
42              call DECOMPOSITION(R); return
43        end

44      :else:  /in this case, f(x) has no 0/
45        if x₀ = 2
46        then
47            case
48              :f(x) = <2,3,2,1>:  split f(x) into
                   <1,1,1,1> & R(x);
49                call DECOMPOSITION (R); return
50              :f(x) = <2,3,2,3>:  split f(x) into
                   <0,1,2,3> & R(x);
51                call DECOMPOSITION (R); return
52              :x_1 > 2, x_2 > 2, x_3 > 2:  split
53                f(x) into <2,2,2,2> & R(x);
54                call DECOMPOSITION; return
55              :else:  split f(x) into the largest
                   two valued C_i(x) [C_i(x) ε D]
                   & R(x);
56                call DECOMPOSITION (R); return
57            end

58        else  / x₀ = 1 or x₀ = 3 /
              if <3,2,1,2> or <3,2,1,3> then
59                split f(x) into <3,2,0,0> & R(x);
60                call DECOMPOSITION (R); return
61              if  {C_i(x),C_j(x)}[C_i(x)+C_j(x) = f(x)]
62              then choose lowest cost pair;
                   return
63              else split f(x) into <x_1,x_1,x_1,x_1>
                   & R(x) such that
                   Min[x_0, x_1, x_2, x_3] = x_1;
64                call DECOMPOSITION (R); return
65      end

end  DECOMPOSITION


procedure MULTIPLE (M)

   case
     :<0,2,0,2>:  call DECOMPOSITION (<0,1,0,1>);
                  attach an inhibit with its input
                  of constant <2> at the end of
                  the string(s); return

     :<0,0,0,2> or <0,0,2,2>:
                  call DECOMPOSITION (M/2);
                  call DECOMPOSITION (M/2); return
```

```
     :else:       call DECOMPOSITION (M/2);
                  attach an inhibit with its input
                  of constant <2> at the end of
                  the string(s); return
   end

end  MULTIPLE
```

Table III. Procedure DECOMPOSITION and MULTIPLE
           for the Cost Table Synthesis of
           4-Valued Unary Functions

Procedure MULTIPLE is called by DECOMPOSITION to synthesize a given function which is a constant times a cost table function. For most functions, the product is obtained by an inhibit at the end of the string(s). However, function $<0,0,0,2>$ and $<0,0,2,2>$ are obtained by adding two identical step functions, $<0,0,0,1>$ and $<0,0,1,1>$, respectively, since this produces a lower cost realization.

The total cost $C\{f\}$ of a target function f is

$$C\{f\} = \sum_{j=1}^{v} C\{S_j\} + (v - 1)C\{A\},$$

where $C\{S_j\}$ is the cost of a cost table function (or a constant times a cost table function) used in the realization of f, v is the number of such functions, and $(v - 1)C\{A\}$ is the cost of adder.

Consider how the algorithm proceeds for the following two examples:

Example 2: $<1,2,3,0>$

Because this is a mixed function with $x_3 = 0$, line 30 of DECOMPOSITION is executed. Further, it corresponds to the case associated with line 37; i.e. it is the sum of cost table functions. There are only seven candidates ($<1,0,0,0>$, $<0,0,1,0>$, $<0,1,0,0>$, $<0,1,1,0>$, $<0,1,2,0>$, $<1,1,0,0>$, and $<1,1,1,0>$) in the cost table for which the target is the sum, and of these only the pair $<0,1,2,0>$ and $<1,1,1,0>$ sums to the function. Fig. 3 shows the decomposition tree. The total cost is 23; 10 + 11 + 2. □
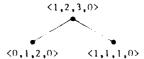
Figure 3. Decomposition of $<1,2,3,0>$.

Example 3: $<3,0,3,2>$

This is also a mixed function and, since $x_2 = 0$, line 22 will be executed. Further, line 27 will be executed, and so the function will be split into $<3,0,0,0>$ and $<0,0,3,2>$. DECOMPOSITION will be called again for each of these functions. $<3,0,0,0>$ will cause line 14 to be executed, splitting this function into $<1,0,0,0>$ and $<2,0,0,0>$. $<0,0,3,2>$ will cause lines 18 and 21 to be executed producing $<0,0,1,0>$ and $<0,0,2,2>$.

MULTIPLE will be called to decompose $\langle 0,0,2,2 \rangle$. Fig. 4 shows the decomposition tree for this function. The total cost is 40; $7 + 7 + 10 + 4 + 4 + (5 - 1) \cdot 2$. □
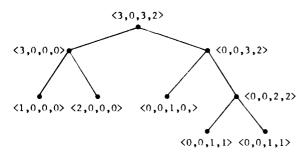
Figure 4. Decomposition Tree of $\langle 3,0,3,2 \rangle$.

Although there are 24 functions in the cost table, 3 ($\langle 3,3,3,3 \rangle$, $\langle 2,3,3,3 \rangle$, and $\langle 0,2,3,3 \rangle$) do not contribute to the decomposition of any other unary function, while 2 ($\langle 1,2,3,3 \rangle$ and $\langle 0,1,3,3 \rangle$) ontribute to the decomposition of only one other function ($\langle 3,2,3,3 \rangle$ and $\langle 2,1,3,3 \rangle$, respectively).

A comparision of the decomposition technique for the 24-entry cost table proposed here and the technique described in Kerkhoff and Robroek [3] and in Robroek [4] using a 45-entry cost table is shown in Lee [9]. For each of the 256 4-valued unary functions, the technique described here produces a decomposition with a cost less than or equal to that produced by the technique described in [3,4]. In 27 or approximately 11% of the functions, there is an improvement†. Further, the realizations produced by the 45-entry cost table approach are achieved by an exhaustive search of all decompositions of a given function, while the realizations produced by the 24-entry table are achieved with considerably less search.

V. THE UNIVERSAL COST TABLE

A further improvement can be obtained by adding to the 24-entry cost table lower cost realizations of functions for which the decomposition method is inefficient. For example, the function of Example 1, $\langle 1,2,3,0 \rangle$ was produced at a cost of 23 by adding two cost table functions. However, another realization exists using a fixed overflow, two inhibits, and an adder, (specifically $\{F(0_x,2,1,0_1,0_2),\ I_1(0_1,0_2,0_d),\ I_1(1,0_2,0_d),$ and $A(0_d^1,0_d^2)\})$ with a cost of only 19.

---

†The cost table technique of [3,4] can be modified so that the cost of all realizations is equal to the cost produced by the technique described here by replacing $F(0_x,1,0_1)$ in the realizations of $\langle 0,1,1,1 \rangle$ and $\langle 0,2,3,3 \rangle$ with $I(1,0_x,0_1)$ and by adding $\langle 0,1,1,2 \rangle$ to the 45-entry cost table. In addition, 22 functions are redundant; they can be removed without increasing the cost of any function. The resulting cost table is the 24-entry table shown here.

Thus, there is a tradeoff between the costs of the funtions produced and the length of the cost table. At one extreme is the cost table of all functions, i.e. the universal cost table. A universal cost table is shown in Lee [9] which improves on 106 functions produced by the technique described here and on 126 functions produced by the technique describe in [3,4]. Because of its large size, the table is not repeated here. However, improvements on the cost table by Lee [9] have been found, and these are shown in Appendix A.

It is interesting to note that a significant portion of the functions in the universal cost table can be eliminated without reducing the average cost of realizations. These are the functions which are the sum of other cost table functions. In the Lee [9] cost table modified by the changes shown in Appendix A, all but 82 functions can be eliminated. Although this 82-entry table produces as efficient realizations as the 256-entry universal cost table, a search for optimum component functions is needed for many functions. Of course, in the universal cost table, such a search is not required; the realization is achieved by a table look-up.

VI. THE SMALLEST COST TABLE

As a basis of comparison, consider the smallest cost table consisting of the four unit functions $\langle 0,0,0,1 \rangle$, $\langle 0,0,1,0 \rangle$, $\langle 0,1,0,0 \rangle$, and $\langle 1,0,0,0 \rangle$. Although, no search is required, the realizations will be of high cost due to the small number of cost table functions. The average cost can be calculated as follows. Over all 256 unary functions, each unit function will be used $6 \cdot 4^3$ (= $0 \cdot 4^3 + 1 \cdot 4^3 + 2 \cdot 4^3 + 3 \cdot 4^3$) times. Since the unit function cost is 7, 10, 10, and 4, respectively, the total cost of the unit functions is $6 \cdot 4^3(7 + 10 + 10 + 4) = 11904$, while the cost of the adders over all unary function is $(4 \cdot 6 \cdot 4^3 - 255)2 = 2562$, for a total cost of 14,466. Dividing by 256 yields 56.51, the average cost. As will be shown later this is considerably larger than the average costs associated with any of the other techniques. The worst case cost, 115, occurs with $\langle 3,3,3,3 \rangle$.

VII. UNIVERSAL UNARY PROGRAMMABLE CIRCUIT (UUPC)

An alternative to the custom design of an unary circuit is a fixed structured circuit in which the function is determined electrically. Fig. 5 shows the configuration. A constant charge $(3Q_E)$ is injected into the root (a) of the tree. The input charge X controls which path the charge flows, while four voltage levels determine the well capacities [1] of the wells associated with nodes d, e, f, and g. If the three charge units from root node a exceeds the capacity of the destination well (d, e, f, or g), the extra charge is discarded into a dump [4]. The single output is the sum of the charges in d, e, f, and g.

This circuit can be designed nicely with a fixed overflow and inhibit circuits, as shown in Fig. 6. The Q_d output wells of inhibit circuits 2
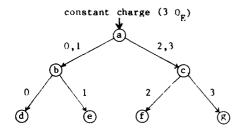
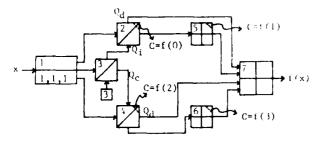Figure 5. The Universal Unary Programmable Circuit.

Figure 6. Decomposition of the UUPC.

Figure 7. A Lower Cost UUPC.

and 4 correspond to nodes $d(f(0))$ and $f(f(2))$ and can be programmed. However, the 0_g output well cannot be programmed, and so programmable wells 5 and 6 must be added for $e(f(1))$ and $g(f(3))$.

The cost of this UUPC implementation is high, 82 (one fixed overflow ~ 4, three inhibits with a 0_i of $3\ 0_E \sim 69$, one constant, one 4-input addition ~ 6, and two programmable wells ~ 2). A cost reduction can be obtained by replacing each of the high cost inhibits with a capacity of $3\ 0_E$ by two inhibits which have a capacity of $1\ 0_E$ and $2\ 0_E$. The cost of the latter is significantly less than the former, so much so that, even though more basic operations are required, the overall cost is less. Fig. 7 show the new UUPC. It is achieved at a cost of 60. With respect to the synthesis of unary functions, the UUPC offers an alternative. Although it is significantly more expensive than any fixed unary function realization by the decomposition method, it is flexible enough to realize any unary function by adjusting four voltage level.

VIII. EXTENDING TO TWO OR MORE VARIABLES

A direct extension of the cost table technique is difficult for $n > 2$ because the large size of such a table could exceed the memory of present day computers. For example, a universal cost table for 4-valued 2-variable functions would have $4.3 \cdot 10^9$ entries. As an alternative, the quaternary or Q-map [4] can be used, in which 4-valued functions are expressed as a composition of one-variable subfunctions. An example is shown in Fig. 8. Here, the four subfunctions can be either the four rows or the four columns. Each subfunction is realized as a unary function by the techniques discussed perviously, while the choice of the subfunction is controlled by the appropriate variable.

Figure 8. Q-map of Function $f(x,y)$.

Example 4: Realization of the Function in Fig. 8.

Consider the two choices for subfunctions, rows of Fig. 8 or columns of Fig. 8. As shown, there is a lower overall cost if the columns are chosen as the subfunctions (the costs listed are from the universal cost table). Thus, we will let x be the controlling variable and synthesize the unary functions as seen in the columns. Fig. 9 shows the decomposition of the function of Fig. 8. The total cost is 171; 74 + 97. □

Fig. 10 and Fig. 11 show the decompositon scheme of the product and carry output, respectively, of the 2-input 4-valued multiplier, using the universal cost table method. It is intresting to note that these multiplier circuits are more cost effective than circuits presented in [4]; the cost of the fomer are 114 (product output) and 32 (carry output), for a total cost of 146, while the costs of the latter are 133 and 50, respectively, for a total cost of 183. It is shown [4] that an implementation using the AllenGivone algebra [3,4] would cost 709, while a method based on a procedure introduced by McCluskey [3] would cost 267.

168
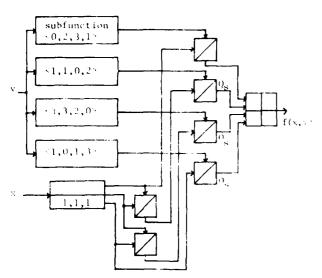
Figure 9. Decompositon of the Two-Variable Functions f(x,y).
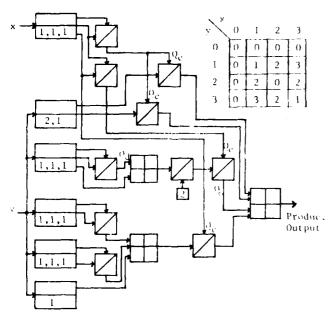
Figure 10. Decomposition of the Product Output of a 2-input 4-valued Mulitplier.

The UUPC implementation can be extended to two or more variables in a straightforward way. Fig. 12 shows a universal binary programmable circuit made from five UUPC's. The total cost of such circuit is 5·60 = 300.

IX. CONCLUDING REMARKS

This paper has focused on various tabular techniques for the implementation of CCD circuits. A tabular method is shown which produces less
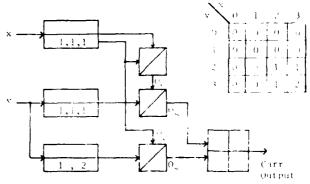
Figure 11. Decomposition of the Carry Output of a 2-Input 4-Valued Multiplier.
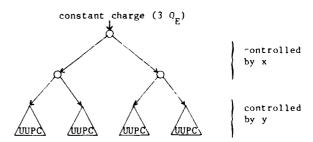
Figure 12. A Universal Binary Programmable Circuit.

expensive 4-valued unary circuits and has almost half the number of entries than does that proposed in [3,4]. A universal cost table is introduced which produces even lower cost realizations at the expense of considerably more entries. In addition, an electrically programmable circuit is introduced in which all unary functions are realized by a single structure of considerably higher cost, but which has the advantage of flexibility; any function can be realized by choosing appropriate voltage levels. Table IV shows a comparison between the four implementations. Although the emphasis has been on 4-valued logic, which is currently realizable [1], the methods can be extended to higher radix.

X. ACKNOWLEDGMENTS

169

	Cost Table of [3, 4]	Improved Cost Table	Universal Cost Table	Smallest Cost Table	Programmable Circuit
number of functions in the cost table	45	24	256	4	0
average cost	21.10	20.72	18.38	56.51	60
maximum cost	47	47	47	115	60
number of functions which are not minimized	126	106	0	246	–
method of realizing a target function	by computer or by hand	by computer or by hand	by computer or by hand (synthesis is a search)	by computer or by hand (synthesis is additions of unit functions)	by electrical programming of well capacities

Table IV. Comparison of the Cost Table and UUPC Implementations of Multiple-Valued Logic Circuits

REFERENCES

[1] H. G. Kerkhoff, M. L. Tervoert, "MultipleValued Logic Charge-Coupled Devices", IEEE Trans. Comput., vol. C-30, No. 9, Sept. 1981, pp. 644-652.

[2] H. G. Kerkhoff, M. L. Tervoert, and H. Tilmans, "Design Considerations and Measurement Results of Multiple-Valued Logic CCD's," Proc. 1981 Int. Symp. Multiple-Valued Logic, Oklahoma City, OK, May 1981, pp. 205-211.

[3] H. G. Kerkhoff, H. A. J. Robroek, "The Logic Design of Multiple-Valued Logic Functions Using Charge-Coupled Devices", Proc. 1982 Int. Symp. Multiple-Valued Logic, Paris, France, May 1982, pp. 35-44.

[4] H. A. J. Robroek, "The Synthesis of MVL-CCD Circuits," M.Sc. report no. 1217.3936, Twente University to Technology, Enschede, Dec. 1981.

[5] O. Ishizuka, "A Consideration for Realizing Unary Functions of a Multi-Valued Variable", Proc. 1982 Int. Symp. Multiple-Valued Logic, May 1982, pp. 89-93.

[6] C. M. Allen, D. D. Givone, "A Minimization Technique For Multiple-Valued Logic System," IEEE Trans. Comput., vol. C-17, Feb. 1968, pp. 182-184.

[7] M. Davio, J-P. Deschamps, "Synthesis of Discrete Functions Using I^2L Technology", IEEE Trans. Comput., vol. C-30, No. 8, Sept. 1981, pp. 653-661.

[8] E. J. McCluskey, "Logic Design of Multi-Valued I^2L Logic Circuit", IEEE Trans. Comput., vol. C-28, No. 8, Aug. 1979, pp. 546-559.

[9] J. K. Lee, "Synthesis Techniques for Four-Valued Logic CCD's Circuit", M.S. Thesis, Northwestern University, Evanston, IL, Aug. 1982.

APPENDIX A

IMPROVEMENTS ON THE REALIZATIONS OF THE UNIVERSAL COST TABLE FUNCTIONS SHOWN IN LEE[9]

Function	Realization	New Cost	Previous Cost
0003	$I(2,0001,0)$; $A(0 ,0001)$*	13	16
0030	$I(2,0010,0^S)$; $A(0^S,0010)$*	19	29
0031	$A(0030, 0_3^S$ of $0010)$ **	21	23
0033	$I(2,0011,0)$; $A(0 ,0011)$*	13	16
0300	$I(2,0100,0^S)$; $A(0^S,0100)$*	19	29
0301	$A(0300, 0_3^S$ of $0100)$**	21	31
0302	$0301 + 0001$	27	31
0303	$I(2,0101,0)$; $A(0_s,0101)$*	21	33
0323	$0200 + 0123$	19	21
0330	$I(2,0110,0)$; $A(0 ,0110)$*	19	29
0331	$A(0330, 0_3^S$ of $0110)$**	21	23
1003	$1000 + 0003$	22	25
1030	$1000 + 0030$	28	38
1031	$1000 + 0031$	30	31
1033	$1000 + 0033$	22	25
1303	$1000 + 0303$	30	34
2003	$2000 + 0003$	22	25
2020	$I(2,0101,0)$*	19	26
2030	$2000 + 0030$	31	38
2031	$2020 + 0011$	25	32
2032	$2020 + 0012$	25	35
2033	$2000 + 0033$	22	25
3003	$I(2,1001,0)$; $A(0_s,1001)$*	22	33
3020	$2020 + 1000$	28	35
3030	$I(2,1010,0)$; $A(0s,1010)$*	28	47
3031	$A(3030, 0_3^S$ of $1010)$**	30	41
3032	$3031 + 0001$	36	40
3033	$I(2,1011,0)$; $A(0_s,1011)$*	22	34
3130	$2020 + 1110$	32	35
3131	$2020 + 1111$	22	29
3303	$I(2,1101,0_s)$; $A(0_s,1101)$*	22	31

* A function of control input Q in an inhibit represents the realization of function itself. The output of that is used as control of the inhibit and as an input of the adder.

** The unused output Q_3 of the fixed overflow in the realization of a function is used as an input of the adder.

Session 4B
Switching Theory

SYNTHESIS METHOD FOR TERNARY LOGIC FUNCTION
BASED ON NAND-TYPE POLYPHECK

Michiaki Yanagita, Naohiro Fukuda, Yoshiaki Miyoshi,
Kyoichi Nakashima and Kazuharu Yamato

Department of Electronics, Himeji Institute of Technology
Himeji, 671-22 JAPAN

Abstract

This paper studies the synthesis of ternary logic functions based on a NAND-type polypheck. The size of term is the arithmetic sum of its truth values over all assignments of variable values. It serves as a measure of complexity.

Introduction

Multiple-valued logic, especially the ternary one, has the following advantages over the binary logic.

(1) data capacity per line,
(2) simplicity of line formation,
(3) speed of data processing,
(4) efficiency of data conveyance.

Some papers about the functional completeness have been reported[1,2,3]. Polypheck is a function which is functionally complete by itself[4]. From both practical and theoretical viewpoints it is interesting to develop a synthesis method based on single devices. In this paper the ternary logic and its utility based on NAND-type polypheck is discussed. The proposed method could serve for the design of ternary logic circuits.

Notation and Definitions

The variables x and y range over the set of truth values $L=(0, 1, 2)$. The logical sum V and logical product $\cdot$ are defined by

$$x \vee y = \max(x, y), \quad x \cdot y = \min(x, y)$$

We introduce a set of 6 prime operators, called literals, on one variable such shown in Table 1 where $\vec{x}$, $\overleftarrow{x}$ are called cyclic, inverse cyclic, respectively. And every ternary logic function has product-of-sum form:

Table 1   A set of 6 prime operators

x	$\vec{x}$	$\overleftarrow{x}$	$\beta_0(x)$	$\beta_1(x)$	$\beta_2(x)$
0	1	2	0	2	2
1	2	0	2	0	2
2	0	1	2	2	0

Table 2   expansion of unary functions

$f(x)$	x			expansion
	0	1	2	
	0	0	0	0
	0	0	1	$\overleftarrow{x} \cdot \overleftrightarrow{x} \cdot \overrightarrow{x}$
	0	0	2	$\overleftrightarrow{x} \cdot \overleftarrow{x} \cdot \overleftrightarrow{x} \cdot x$
	0	1	0	$x \cdot \vec{x}$
	0	1	1	$x \cdot (\overleftarrow{x} \cdot \overleftrightarrow{x} \vee \overleftarrow{x})$
	0	1	2	x
	0	2	0	$\vec{x} \cdot \overleftrightarrow{x} \cdot \overleftarrow{x}$
	0	2	1	$\vec{x} \cdot \overleftarrow{x} \cdot (x \cdot \overleftarrow{x} \vee \overleftarrow{x})$
	0	2	2	$\overleftrightarrow{x} \cdot \overleftarrow{x}$
	1	0	0	$\vec{x} \cdot \overleftrightarrow{x} \cdot \overleftarrow{x}$
	1	0	1	$\overleftarrow{x} \cdot (\overleftrightarrow{x} \cdot \overleftarrow{x} \vee \vec{x})$
	1	0	2	$(\overleftrightarrow{x} \cdot \overleftarrow{x} \vee \vec{x}) \cdot \overleftrightarrow{x} \cdot x$
	1	1	0	$\vec{x} \cdot (\overleftrightarrow{x} \cdot \overleftarrow{x} \vee \vec{x})$
	1	1	1	1
	1	1	2	$(\overleftrightarrow{x} \cdot \overleftarrow{x} \vee \vec{x}) \cdot (\overleftrightarrow{x} \cdot x \vee x)$
	1	2	0	$\vec{x}$
	1	2	1	$(\overleftarrow{x} \cdot \overleftarrow{x} \vee \vec{x}) \cdot (x \cdot \overleftarrow{x} \vee \overleftarrow{x})$
	1	2	2	$\overleftrightarrow{x} \cdot \overleftarrow{x} \vee \vec{x}$
	2	0	0	$\overleftarrow{x} \cdot \overleftarrow{x} \cdot \overleftrightarrow{x}$
	2	0	1	$\overleftarrow{x}$
	2	0	2	$\overleftrightarrow{x} \cdot x$
	2	1	0	$(\overleftrightarrow{x} \cdot x \vee x) \cdot \overleftarrow{x} \cdot \overleftarrow{x}$
	2	1	1	$(\overleftarrow{x} \cdot \overleftrightarrow{x} \vee \overleftarrow{x}) \cdot (\overleftrightarrow{x} \cdot x \vee x)$
	2	1	2	$x \cdot \overleftarrow{x} \cdot x$
	2	2	0	$x \cdot \overleftarrow{x}$
	2	2	1	$x \cdot \overleftarrow{x} \vee \overleftarrow{x}$
	2	2	2	2

$$f(x_1, \ldots, x_n)$$

$$= \prod_{(a_1, \cdots, a_n)} \left\{ f(a_1, \ldots, a_n) \vee x_1^{\overleftarrow{a_1}} \vee \beta_1^{a_1}(x_1) \vee \ldots \vee x_n^{\overleftarrow{a_n}} \vee \beta_n^{a_n}(x_n) \right\} \tag{1}$$

where $x^0 = x$, $x^1 = \vec{x}$, $x^2 = \overleftarrow{x}$ and $\overleftarrow{a} = (\overleftarrow{a})$ and $\overline{a} = 2 - a$.

The three 0-consensus functions $\beta^0(x)$, $\beta^1(x)$, $\beta^2(x)$ are represented as follows.

$$\beta^0(x) = \overleftarrow{x} \cdot \overrightarrow{x}, \quad \beta^1(x) = \overleftarrow{\overrightarrow{x \cdot x}}, \quad \beta^2(x) = x \cdot \overleftarrow{x}$$

Only Eq.(1) is the form which includes all 6 prime operators. The unary functions are expanded as in Table 2.

Definition 1: The logical sum $E(\underline{x})$, $(\underline{x}) = (x_1, x_2, \ldots, x_n)$, which often abbreviated as E, of some distinct literals is a term of the order denoted by $O(E)$ which is the number of literals included in E. The total number of literals in $f(\underline{x})$, which often abbreviated as f is said to be the degree of f and denoted by $D(f)$.

Definition 2[5]: We also treat constants " 0 ", " 1 ", " 2 " as terms, and define

$$O(0) = 0, \qquad O(2) = 2n+1$$

We do not define the order of constant " 1 " because it can be represented by the product-of-sum form of at most 2n-order terms.

Size of Term

Definition 3: The size $S(E)$ of a term E is defined as the pair $(a+2b)[b]$ where a and b are the number of truth values of E, 1 and 2.

Lemma 1: Consider the size of $E = \overleftarrow{x}^a \vee \overleftarrow{x}^b \cdot x^c$ (a, b, $c \in L$).

(i) If $a = b = c$, $E = \overleftarrow{x}^a$. Then,

$$S(E) = 3[1].$$

(ii) If $a \neq b = c$, $E = \overleftarrow{x}^a \vee \overleftarrow{x}^b$. Then,

$$S(E) = 5[2].$$

(iii) In other cases,

$$S(E) = \begin{cases} 6[3] & \text{if } a \oplus b \oplus c = 0 \quad (2) \\ 4[2] & \text{if } a \oplus b \oplus c = 1 \quad (3) \\ 5[2] & \text{if } a \oplus b \oplus c = 2 \quad (4) \end{cases}$$

where $\oplus$ denotes the mod.3 sum.

From Eq.(1), any term E can be represented as follows.

$$E = \overleftarrow{x_1}^{a_1} \vee \overleftarrow{x_1}^{a_2} \cdot x_1^{a_3} \vee \overleftarrow{x_2}^{a_4} \vee \overleftarrow{x_2}^{a_5} \cdot x_2^{a_6} \cdots$$
$$\cdots \vee \overleftarrow{x_n}^{a_{3n-2}} \vee \overleftarrow{x_n}^{a_{3n-1}} \cdot x_n^{a_{3n}}$$
$$a_{3i-2}, a_{3i-1}, a_{3i} \in L, \quad 1 \leq i \leq n \quad (5)$$

and define the parameter vector P as follows.

$$P = (a_1 \oplus a_2 \oplus a_3, a_4 \oplus a_5 \oplus a_6, \cdots, a_{3n-2} \oplus a_{3n-1} \oplus a_{3n})$$

Lemma 2: Consider the size of 2n-order term E given by Eq.(5) where $a_{3i-1} \neq a_{3i}$ ($1 \leq i \leq n$). Then,

$$S(E) = \begin{cases} (2 \times 3^n)[3^n] & \text{if } p \neq 0, \ p+q+r = n \\ (2 \times 3^n - 1)[3^n - 1] & \text{if } p = 0, \ r \neq 0, \ q+r = n \end{cases}$$

$$\left| \{2 \times (3^n - 1)\}[3^n - 1] \right.$$
$$\text{if } p = r = 0, \ q = n$$

where p, q, r are the numbers of element " 0 ", " 1 ", " 2 ", respectively, in P.

Proof: Define the following term E_1.

$$E_1 = \overleftarrow{x_{i+1}}^{a_{3(i+1)-2}} \vee \overleftarrow{x_{i+1}}^{a_{3(i+1)-1}} \cdot x_{i+1}^{a_{3(i+1)}} \vee \cdots$$
$$\cdots \vee \overleftarrow{x_n}^{a_{3n-2}} \vee \overleftarrow{x_n}^{a_{3n-1}} \cdot x_n^{a_{3n}}$$

1) If $P = (2, 2, \ldots, 2)$, then

$$S(E) = S(\overleftarrow{x_1}^{a_1} \vee \overleftarrow{x_1}^{a_2} \cdot x_1^{a_3} \vee E_1)$$
$$= S(2 \vee E_1) + S(2 \vee E_1) + S(1 \vee E_1)$$
$$\text{(from Eq.(4))}$$
$$= 2 \times 3^{n-1} + 2 \times 3^{n-1} + S(1 \vee E_1)$$
$$= 2 \times 3^n - 1$$

where + denotes the arithmetic sum. In this case, it is clear from the process of this proof

that this term E takes 3^{n-1} truth values " 2 ". Therefore, from the Definition 3, $S(E) = 2 \times 3^n - 1 [3^n - 1]$.

2) If $P = (1, 1, \ldots, 1)$, then from Eq.(3),

$$S(E) = \{2 \times (3^n - 1)\}[3^n - 1]$$

3) If $P = (0, 0, \ldots, 0)$, then from Eq.(2),

$$S(E) = (2 \times 3^n)[3^n]$$

4) If $r_i \in \{1, 2\}$ for all i ($q \neq n$, $r \neq n$, $q + r = n$), then

$$E = \hat{E}_q \vee E_q$$

where suffix numbers of x, a, are renumbered such that the mod.3 sum of the superscripts of each variable in $\hat{E}_q$ is 1 and that in E_q is 2, i.e.,

$$a_1 \oplus a_2 \oplus a_3 = a_4 \oplus a_5 \oplus a_6 = \ldots$$
$$\ldots = a_{3q-2} \oplus a_{3q-1} \oplus a_{3q} = 1$$
$$a_{3q+1} \oplus a_{3q+2} \oplus a_{3q+3} = a_{3q+4} \oplus a_{3q+5} \oplus a_{3q+6} = \ldots$$
$$\ldots = a_{3n-2} \oplus a_{3n-1} \oplus a_{3n} = 2$$

$$\hat{E}_q = \overleftarrow{x_1}^{a_1} \vee \overleftarrow{x_1}^{a_2} \cdot x_1^{a_3} \vee \overleftarrow{x_2}^{a_4} \vee \overleftarrow{x_2}^{a_5} \cdot x_2^{a_6} \vee \ldots$$
$$\ldots \vee \overleftarrow{x_q}^{a_{3q-2}} \vee \overleftarrow{x_q}^{a_{3q-1}} \cdot x_q^{a_{3q}}$$

$$E_q = \overleftarrow{x_{q+1}}^{a_{3(q+1)-2}} \vee \overleftarrow{x_{q+1}}^{a_{3(q+1)-1}} \cdot x_{q+1}^{a_{3(q+1)}} \vee \ldots$$
$$\ldots \vee \overleftarrow{x_n}^{a_{3n-2}} \vee \overleftarrow{x_n}^{a_{3n-1}} \cdot x_n^{a_{3n}}$$

Therefore,

$$S(E) = S(\overleftarrow{x_1}^{a_1} \vee \overleftarrow{x_1}^{a_2} \cdot x_1^{a_3} \vee E_1)$$
$$= S(2 \vee E_1) + S(2 \vee E_1) + S(0 \vee E_1)$$
$$\text{(from Eq.(3))}$$

. . . .

173

$$= 2^2 \times (3^{n-1} + 3^{n-2} + \ldots + 3^{n-q+1} + 3^{n-q}) + S(E_q)$$

$$= 2^2 \times (3^{n-1} + 3^{n-2} + \ldots + 3^{n-q+1} + 3^{n-q})$$
$$+ S(2 \vee E_{q+1}) + S(2 \vee E_{q+1}) + S(1 \vee E_{q+1})$$
$$\text{(from Eq.(4))}$$

$$= 2 \times 3^n - 1 \ [\ 3^n - 1]$$

5) If $P_i \in \{0, 1\}$ for all i ($P \neq 0$, $P + q = n$), then
$$S(E) = 2 \times 3^n \ [\ 3^n \ ]$$

6) If $p_i \in \{0, 2\}$ for all i ($p \neq 0$, $p + r = n$), then
$$S(E) = 2 \times 3^n \ [\ 3^n \ ]$$

7) If $P_i \in \{0, 1, 2\}$ for all i ($p \neq 0$, $p + q + r = n$), then
$$S(E) = 2 \times 3^n \ [\ 3^n \ ]$$

This concludes the proof.

Lemma 3: Consider the size of $(2n-j)$-order term E' which is an n-variable function ($1 \leq j \leq n$). There are two forms of E'. One is the form which consists of literals of n variables. Another is the form which consists of literals of $(n-i)$ variables ($1 \leq i \leq n$, $i \leq j$). In the former case, E' can be represented as follows.

$$E' = x_1^{\overleftarrow{a_3}} \vee x_2^{\overleftarrow{a_6}} \vee \ldots \vee x_i^{\overleftarrow{a_{3i}}} \vee x_{i+1}^{\overleftarrow{a_{3(i+1)-1}}} \cdot x_{i+1}^{a_{3(i+1)}} \vee \ldots$$
$$\ldots \vee x_j^{a_{3j-1}} \cdot x_j^{a_{3j}} \vee E_j$$

where $a_{3k-1} \neq a_{3k}$ ($i+1 \leq k \leq n$, $0 \leq i \leq j$). The parameter vector P' of E' is that of E_j. Then,

$$S(E') = \begin{cases} (2 \times 3^n) \ [\ 3^n \ ] \\ \qquad \text{if } p' \neq 0, \ p' + q' + r' = n-j \\ (2 \times 3^n - 2^i) \ [\ 3^n - 2^i] \\ \qquad \text{if } p' = 0, \ r' \neq 0, \ q' + r' = n-j \\ (2 \times 3^n - 2^i - 1) [\ 3^n - 2^i] \\ \qquad \text{if } p' = r' = 0, \ q' = n-j \end{cases}$$

where p', q', r' are the number of element " 0 ", " 1 ", " 2 ", respectively, in P'.

In the latter case, E' can be represented as follows.

$$E' = x_{i+1}^{\overleftarrow{a_{3(i+1)}}} \vee x_{i+2}^{\overleftarrow{a_{3(i+2)}}} \vee \ldots \vee x_l^{\overleftarrow{a_{3l}}} \vee x_{l+1}^{a_{3(l+1)-1}} \cdot x_{l+1}^{a_{3(l+1)}} \vee \ldots$$
$$\ldots \vee x_{j-i}^{a_{3(j-i)-1}} \cdot x_{j-i}^{a_{3(j-i)}} \vee E_{j-i}$$

where $a_{3k-1} \neq a_{3k}$ ($l+1 \leq k \leq n$, $i \leq l \leq j-i$). The parameter vector P' of E' is that of E_{j-i}. Then,

$$S(E') = \begin{cases} (2 \times 3^n) \ [\ 3^n \ ] \\ \qquad \text{if } p' \neq 0, \ p' + q' + r' = n-j \\ 3^i \times (2 \times 3^{n-i} - 2^{l-i}) \ [\ 3^i \times (3^{n-i} - 2^{l-i})] \\ \qquad \text{if } p' = 0, \ r' \neq 0, \ q' + r' = n-j \\ 3^i \times (2 \times 3^{n-i} - 2^{l-i} - 1) \ [\ 3^i \times (3^{n-i} - 2^{l-i})] \\ \qquad \text{if } p' = r' = 0, \ q' = n-j \end{cases}$$

where P', q', r' are the number of element " 0 ", " 1 ", " 2 ", respectively, in P'.

From Lemmas 2 and 3, we can find out that the size of a term in a canonical form tends to decrease with the decrease of its order.

Subterm and Implicant

Definition 4: The ternary n-variable function f implies the ternary n-variable function g (notation: $f \leq g$) if for every $(a_1, a_2, \ldots, a_n)$

$$f(a_1, a_2, \ldots, a_n) \leq g(a_1, a_2, \ldots, a_n)$$

If f doesn't imply g, then it is denoted as $f \nleq g$.

Definition 5: A term e is said to be the subterm of a term E if e is the remainder after the removal of at least one literal from E.

Lemma 4: If e is the subterm of E, then the order of e is lower than that of E, i.e. $O(e) < O(E)$ and e implies E, $e \leq E$. From Definition 2 constants " 0 ", " 1 ", " 2 " also satisfy this lemma.

Definition 6: A term E is said to be the implicant of a ternary logic function f if E satisfies the following two conditions.

1) $E \geq f$ (6)
2) $e \ngeq f$ for any subterm e of E.   (7)

Definition 7: A simplest form of a given n ternary logic function is a form minimizing the degree of the function.

Lemma 5: If a product-of-sum form of f is simplest, then every term is an implicant of f and each term is equal to $f < 2$ for at least one assignment of variable values.

Proof: Assume that the simplest form of f is

$$f = E_1 \cdot E_2 \cdot \ldots \cdot E_k \qquad (8)$$

Clearly Eq.(6) is satisfied. Suppose, for a subterm e_1 of E_1

$$e_1 \geq f$$

Then,

$$f = E_1 \cdot E_2 \cdot \ldots \cdot E_k$$
$$\geq e_1 \cdot E_2 \cdot \ldots \cdot E_k$$
$$\geq f \cdot f \cdot \ldots \cdot f = f$$

Therefore,

$$f = e_1 \cdot E_2 \cdot \ldots \cdot E_k \qquad (9)$$

The degree of (9) is lower than that of (8). This contradicts the assumption. This completes the proof of the first half. The latter half is self-evident. This finished the proof.

Definition 8 : Let the terms $E_1, E_2, \ldots, E_k$ be all implicants of f. If an implicant E_i satisfies the following equation,

$$E_i \geq E_1 \cdot E_2 \cdot \ldots \cdot E_{i-1} \cdot E_{i+1} \cdot \ldots \cdot E_k \qquad (10)$$

then f can be realized without E_i. An implicant which doesn't satisfy the Eq.(10) is called an essential implicant.

174

Simplification Algorithm

Algorithm for Determination of Implicant

If 2n-order term F is an implicant, then for its (2n-1)-order subterm e, $E \supseteq e \not\equiv f$, and its (2n-k)-order subterm e' is also the subterm of (2n-k+1)-order term e'', $2 \leqq k \leqq 2n$. Then, $2n-1 \geqq 2n-k+1 \geqq 1$. Therefore, when we want to check if a term E whose order takes from 1 to 2n is an implicant, we only examine the conditions of the implicant for the subterms with the order by one lower than E. A term is expressed as the logical sum of several literals. Therefore, the algorithm for the determination of implicants is as follows.

Step 1: Make a term E by taking one by one out of 2n sets $\{\vec{\pi}_k, \overrightarrow{\pi_k}, \overleftarrow{\pi_k}, 0\}$, $\{\vec{\pi}_k, \overrightarrow{\pi_k}, \overleftarrow{\pi_k}, \overleftrightarrow{\pi_k \cdot \pi_k}, \overleftrightarrow{\pi_k \cdot \pi_k}, \overleftarrow{\pi_k \cdot \pi_k}, 0\}$, k=1, 2, ..., n, and combine them by logical sum.

Step 2: Examine whether or not the term E satisfies the following condition.

$$E \supseteq f$$

where the equality must hold for at least one assignment of variable values for which $f = 0$ or 1. If not, return to step 1 and make a new term. If satisfied, proceed to step 3.

Step 3: Examine whether or not the following condition is satisfied for the subterms e with the order by one lower than E.

$$e \not\supseteq f$$

If not, return to step 1 and make a new term. If satisfied, proceed to step 4.

Step 4: The term obtained in above steps is an implicant. So write it out. Repeat above steps until no more combination of literals is found. Then this algorithm is terminated.

Simplification Algorithm

Assume that all implicants of f have been obtained.

Step 1: Construct a table relating the truth values of f to the truth values of each implicant such as Table 3.

Step 2: Encircle the point at where $E = f < 2$.

Step 3: Determine essential implicants.

Step 4: Determine the implicants which realize some truth values " 0 " of f that are not realized by the product-of-sum form of essential implicants from lower order terms.

Step 5: Determine the implicants which realize the remaining truth values of f from lower order terms.

The product-of-sum form of implicants obtained in above steps is the simplest.

Synthesis Algorithm

We start the synthesis algorithm with the simplest form of a function f. Assume that the simplest form of f consists of k implicants, F_1, F_2, ..., F_k.

Step 1: Operate cyclic or inverse cyclic operator upon each term and determine their sizes $S(F_i^{\,i})$,

$i \in L$ ($1 \leqq i \leqq k$).

Step 2: For the term $E_i^{\,i^*}$, $\alpha^* \in L$, having the smallest size (in the case having two or three terms with the smallest size, for the one which has the greater number of truth value " 0 "), determine its implicants. If both the size and the number of truth value " 0 " are the same, then determine the implicants of $\overrightarrow{E_i}$.

Step 3: Determine the simplest form of $E_i^{\,\alpha^*}$.

Step 4: Assume that the simplest form of $E_i^{\,\alpha^*}$ is as follows.

$$F_i^{\,i^*} = I_1 \cdot I_2 \cdot \ \cdots \ \cdot I_m$$

Then,

$$E_i = \left\{ \ (I_1 \cdot I_2 \cdot \ \cdots \ \cdot I_m)^{\,i^*} \ \right\}^{i^*}$$

Repeat above steps for I_i, $1 \leqq i \leqq m$, until every implicant can be expressed as the logical product of some literals.

From above steps, any ternary logic function can be synthsized based on NAND-type polypheck.

Example: Consider the function $f(x, y)$, ternary half-adder, given in Table 3. First from the algorithm for the determination of implicant, we have 15 implicants as shown in Table 3. From the simplification algolithm, the simplest form of $f(x, y)$ is

$$f(x, y) = (\ x \vee \overleftrightarrow{y \cdot y}\ ) \cdot (\ \vec{x} \vee \overleftarrow{y \cdot y}\ ) \cdot (\ \overleftarrow{x} \vee \overleftarrow{y \cdot y}\ )$$

$$= (\ \overleftrightarrow{x \cdot x} \vee y\ ) \cdot (\ \overleftrightarrow{x \cdot x} \vee \vec{y}\ ) \cdot (\ \overleftarrow{x \cdot x} \vee y\ )$$

Consider the former case. Define the following terms.

$$E_1 = x \vee \overleftrightarrow{y \cdot y}\ , \quad E_2 = \vec{x} \vee \overleftarrow{y \cdot y}\ , \quad E_3 = \overleftarrow{x} \vee \overleftrightarrow{y \cdot y}$$

From step 1 in the synthesis algorithm,

$$S(E_1) = 15[\ 7\ ], \quad S(\vec{E_1}) = 3[\ 1\ ], \quad S(\overleftarrow{E_1}) = 9[\ 1\ ]$$

$$S(E_2) = 15[\ 7\ ], \quad S(\vec{E_2}) = 3[\ 1\ ], \quad S(\overleftarrow{E_2}) = 9[\ 1\ ]$$

$$S(E_3) = 15[\ 7\ ], \quad S(\vec{E_3}) = 3[\ 1\ ], \quad S(\overleftarrow{E_3}) = 9[\ 1\ ]$$

Therefore, determine the implicants of $\vec{E_1}$, $\vec{E_2}$, $\vec{E_3}$. We have 6 implicants of $\vec{E_1}$, $\vec{E_2}$, $\vec{E_3}$, respectively. The simplest form of them are

$$\vec{F_1} = \vec{x} \cdot \vec{y} \cdot \overleftarrow{y \cdot y}$$

$$\vec{F_2} = \overleftarrow{x} \cdot \vec{y} \cdot \overleftrightarrow{y \cdot y}$$

$$\vec{F_3} = x \cdot y \cdot \overleftrightarrow{y \cdot y}$$

Therefore,

$$f(x, y) = \overline{\vec{x} \cdot \overleftarrow{y} \cdot y \cdot y} \cdot \overline{x \cdot \overleftarrow{y} \cdot \vec{y} \cdot y} \cdot \overline{x \cdot y \cdot \vec{y} \cdot y}$$

175

Table 3   Implicants of $f(x, y)$

x	0	0	0	1	1	1	2	2	2
y	0	1	2	0	1	2	0	1	2
$f(x, y)$	0	1	2	1	2	0	2	0	1
$x \vee \overrightarrow{x \vee y}$	1	①	2	2	2	2	2	2	2
$x \vee \overleftarrow{x \vee y}$	2	2	2	①	2	1	2	2	2
$x \vee \overleftrightarrow{y \cdot y}$	⓪	2	2	①	2	2	2	2	2
$\overrightarrow{x} \vee \overleftarrow{x \vee y}$	2	2	2	2	2	2	2	1	①
$\overrightarrow{x} \vee y \vee \overleftarrow{y}$	2	①	2	2	2	2	2	1	2
$\overrightarrow{x} \vee \overleftrightarrow{y \cdot y}$	2	①	2	2	2	2	2	⓪	2
$\overleftarrow{x} \vee \overrightarrow{y} \vee y$	2	2	2	2	2	1	2	2	①
$\overleftarrow{x} \vee \overleftarrow{y \cdot y}$	2	2	2	2	2	⓪	2	2	①
$\overrightarrow{x \cdot x} \vee y$	⓪	①	2	2	2	2	2	2	2
$\overrightarrow{x \cdot x} \vee \overleftrightarrow{y \cdot y}$	⓪	2	2	2	2	2	2	2	2
$x \cdot \overleftarrow{x} \vee y$	2	2	2	2	2	2	2	⓪	①
$x \cdot \overleftarrow{x} \vee \overleftrightarrow{y \cdot y}$	2	2	2	2	2	2	2	⓪	2
$\overleftrightarrow{x \cdot x} \vee \overrightarrow{y}$	2	2	2	①	2	⓪	2	2	2
$\overleftrightarrow{x \cdot x} \vee y \cdot \overleftarrow{y}$	2	2	2	2	2	⓪	2	2	2
$x \vee y \vee \overrightarrow{y}$	1	2	2	①	2	2	2	2	2

This function requires 13 NAND-type gates to implement the function as shown in Fig. 1. If we apply the synthesis method to $\overline{E}_1$, $\overline{E}_2$, $\overline{E}_3$, then $f(x, y)$ requires 28 NAND-type gates. The original function requires 82 NAND-type gates.
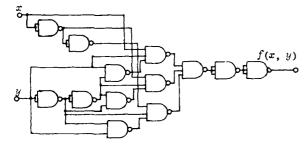

Fig.1   diagram of $f(x, y)$

Utility of NAND-type polypheck

Definition 9: A function is linear whose modular expansion form contains less than first order terms. A function is symmetric if $f(x, y) = f(y, x)$. $f(x, x)$ is denoted by $F(x)$. A 2-variable ternary polypheck $f(x, y)$ is called LSP if $f(x, y)$ is symmetric and $F(x)$ is linear.
Lemma 6: $f(x, y)$ is LSP if and only if its modular expansion form

$$f(x, y) = \alpha_0 \oplus \alpha_1 x \oplus \alpha_2 y \oplus \alpha_3 xy \oplus \alpha_4 x^2 \\ \oplus \alpha_5 y^2 \oplus \alpha_6 x^2 y \oplus \alpha_7 xy^2 \oplus \alpha_8 x^2 y^2$$

satisfies the following conditions.

$$\alpha_1 = \alpha_2, \quad \alpha_4 = \alpha_5, \quad \alpha_6 = \alpha_7$$

$$\alpha_3 \oplus 2\alpha_4 \oplus \alpha_8 = 0$$
$$\alpha_0 = 1 \text{ or } 2, \quad \alpha_1 \oplus \alpha_6 = 2 \quad (\alpha_i \in L, \ 0 \leq i \leq 8)$$

The proof of this lemma is omitted.

From lemma 6, there are 54 LSPs. Out of these 54 LSPs, we select polypheck suitable for the product-of-sum form. Then, we set following condition.

It is easy to generate 0-consensus functions and logical product from the polypheck.

We try to synthesize these functions with the polypheck $f(x, y)$ within the limits of 4 stages. The result is shown in Table 4 where

$$f_1(x, y) = 1 \oplus x y \oplus 2x^2 y \oplus 2x y^2 \oplus 2x^2 y^2 = \overrightarrow{x \cdot y}$$

$$f_2(x, y) = 2 \oplus x y \oplus 2x^2 y \oplus 2x y^2 \oplus 2x^2 y^2 = \overleftarrow{x \cdot y}$$

Other 52 LSPs cannot generate all four functions, $\beta^i(x)$ and logical product within the limits. We prefer $f_2(x, y)$ to $f_1(x, y)$ because $f_2(x, y)$ requires less gates to generate $\beta^0(x)$ and $\beta^2(x)$.

Table 4   selection of polypheck

	1 stage	2 stages	3 stages	4 stages
$\beta^0(x)$			f_2	f_1
$\beta^1(x)$			$f_1 \ f_2$	
$\beta^2(x)$		f_2		f_1
logical product			$f_1 \ f_2$	

Conclusion

The proposed method is simple, systematic and familiar to us because the simplification method is similar to that in binary logic. It is also easy to apply many variables. In this sense, the proposed method is the best.

Reference
(1) E.L. Post,:"Introduction to a general theory of elementary propositions", Amer. J. Math., 43, p.163 (1921).
(2) N.M. Martin,:"The Sheffer function of 3-valued logic", The Journal of Symbolic logic, 19, p.45 (March 1954).
(3) D.L. Webb,:"Generation of any n-valued logic by one binary operation", Proc. Nat. Acad. Sci., 21, p.252 (1935).
(4) Tanaka, S. and Tahara, M.,:"Functional completeness and polychecks in 3-valued logic", (in Japanese), Trans., IECE Japan, 53-C,2,p.111 (1970-02).
(5) Nozaki, A.,:"Switching theory"(in Japanese), Kyoritsu Syuppan.

VECTOR EXPANSION TRANSFORMATION OF LOGIC ALGEBRA

by Liu Zhimo and Yuan Youguang

Wanzhong Machinery Plant, Sichuan, China
Wuhan Digital Engineering Institute, Hubei, China

Abstract

We analyse the mathematical structure of the vector expression of star algorithm here, where the concept of vector expansion transformation is presented. We also describe this transformation which can be generalized to the general Boolean algebra as well as other logic systems. Virtually we demonstrate that the operation problems of general logic value can be transormed into those of binary Boolean expression and advance calculation formulas of transformation under various situations. It is proved that this method is of certain significance in switching theory and in practical applications.

Introduction

In the fault diagnosis of logic networks, the multivalued logic method is widely used[1-4], which, starting from different point of view, considered the compatibility with logic variable valuing, as a result of those achievements the calculation process is simplified. Especially in [1] the vector expression is presented, which with obvious advantages transforms the four-valued logic calculation problems into those of binary Boolean expression. In this paper we are going to start from the analysis of star algorithm in [1] to explicit the essense of the vector expression and generalize this method into more general situations. It is proved that the result of this paper is more concrete both in theory and application.

Vector Expansion Transformation of Four-Valued Boolean Algebra

In [1] the vector calculation problem of four-valued Boolean algebra $B_4 = B_2 B_2$ has been studied. In this section let us first list some of the results in [1] and then make further discussion theoretically.

Let the elements set of B_4 be $\{\theta, I, D, \overline{D}\}$, as shown in Fig. 1(a), where $\theta = (0,0)$ being minimal element, $I = (1,1)$ being maximal element, $D = (1,0)$, $\overline{D} = (0,1)$ being respectively their atoms.

Theorem 1. For any variable x in B_4, to be corresponding to which, choose four proposition variables x^0, x^1, x^*, $\overline{x}^*$ in B_2 such that
$$x = \theta \text{ iff } x^0 = 1 \text{ and } x^1 = x^* = \overline{x}^* = 0,$$
$$x = I \text{ iff } x^1 = 1 \text{ and } x^0 = x^* = \overline{x}^* = 0,$$
$$x = D \text{ iff } x^* = 1 \text{ and } x^0 = x^1 = \overline{x}^* = 0,$$
$$x = \overline{D} \text{ iff } \overline{x}^* = 1 \text{ and } x^0 = x^1 = x^* = 0,$$
$$(1)$$
then
$$x = x^0 \theta + x^1 I + x^* D + \overline{x}^* \overline{D} \qquad (2)$$
satisfies
$$x^0 + x^1 + x^* + \overline{x}^* = 1 \qquad (3)$$
$$x^i \cdot x^j = 0 \quad i, j \in \{0, 1, *, -*\}, \quad i \neq j \qquad (4)$$
Eq.(2) is said to be the vector expression of x and x is called vector. So x^0, x^1, x^*, $\overline{x}^*$ are called components of x.

Definition 1. Set the vector expression of $x \in B_4$ be
$$x = a \cdot \theta + b \cdot I + c \cdot D + d \cdot \overline{D}, \quad a, b, c, d \in B_2$$
if a, b, c, d satisfy Eqs.(3) and (4), then we call this vector expression standardized vector expression.

Lemma 1. Set any vector expression x be
$$a = a \cdot \theta + b \cdot I + c \cdot D + d \cdot \overline{D}, \quad a, b, c, d \in B_2$$
then this one can be transformed into the standardized vector expression
$$x = x^0 \cdot \theta + x^1 \cdot I + x^* \cdot D + \overline{x}^* \cdot \overline{D},$$
and there exists
$$x^0 = \overline{bcd}, \quad x^1 = b + cd,$$
$$x^* = b\overline{c}d, \quad \overline{x}^* = \overline{b}cd. \qquad (5)$$

Let
$$x = x^0 \cdot \theta + x^1 \cdot I + x^* \cdot D + \overline{x}^* \cdot D,$$
$$y = y^0 \cdot \theta + y^1 \cdot D + y^* \cdot D + \overline{y}^* \cdot \overline{D}$$
be the standardized vector expressions of x, y, the following recurrence formulas of basic logic operation is obtained in [1]:
$$(\overline{x})^0 = x^1, \quad (\overline{x})^1 = x^0, \qquad (6)$$
$$(\overline{x})^* = \overline{x}^*, \quad (\overline{x})^* = x^*,$$
$$(x \cdot y)^0 = x^0 + y^0 = x^* \overline{y}^* + \overline{x}^* y^*, \quad (x \cdot y)^1 = x^1 \cdot y^1$$
$$(x \cdot y)^* = x^* y^1 + x^1 y^* + x^* y^*, \qquad (7)$$
$$\overline{(x \cdot y)}^* = \overline{x}^* y^1 + x^1 \overline{y}^* + \overline{x}^* \overline{y}^*$$

177

$(x+y)^0=x^0y^0, (x+y)^1=x^1+y^1+x^*\overline{y}^*+\overline{x}^*y^*$

$(x+y)^*=x^*y^0+x^0y^*+x^*y^*,$ (8)

$\overline{(x+y)}^*=\overline{x}^*y^0+x^0\overline{y}^*+\overline{x}^*\overline{y}^*$

Eqs.(6), (7) and (8) are called component recurrence formulas of four-valued variables, they show the combination situation when logic gate output is taken by certain definite value, its input may also take value.

Theorem 2. Let $F(x_1, \ldots, x_n)$ be any Boolean function of $B_4^n \to B_4$, then F can be expressed by the vector expression, and its components can be calculated recurrently gate by gate according to Eqs. (6), (7) and (8) and structure rule of the function.

In the fault diagnosis of logic networks, by the stipulation of θ, I, D, $\overline{D}$, we could make comprehensive analysis of fault-free circuit and faulty circuit, the component F^* and $\overline{F}^*$ expansion of F, therefore, could be used to generate the tests, and F^0, F^1 can be used to analyse the problems of fault-tolerence, etc. In [1] the recurrent calculation method according to Eqs. (6), (7) and (8) is called star algorithm.

Definition 2. Let $x \in B_4$, define $\lfloor x = x^1 + x^*$, $x \rfloor = x^1 + \overline{x}^* \lfloor x$ is called the anterior component of x, and $x \rfloor$ is called the posterior component of x. Obviously

$$\lfloor \overline{x} = \overline{(\lfloor x)} = x^0 + \overline{x}^*, \quad \overline{x} \rfloor = \overline{(x \rfloor)} = x^0 + x^*.$$

Theorem 3. Let $F = F(x_1, \ldots, x_n)$, then

$$\lfloor F = F(\lfloor x_1, \ldots, \lfloor x_n), \quad F \rfloor = F(x_1 \rfloor, \ldots, x_n \rfloor). \quad (9)$$

Theorem 4. Let $x = x_1 \ldots x_i \ldots x_m$, then

$$x^* = \sum_{i=1}^{m} (x_i^* \prod_{\substack{j=1 \\ j \neq i}}^{m} \lfloor x_j) \quad \overline{x}^* = \sum_{i=1}^{m} (\overline{x}_i^* \prod_{\substack{j=1 \\ j \neq i}}^{m} x_j \rfloor)$$

It is not difficult to generalize this into other types of gates.

Since $\lfloor x$, $x \rfloor$ show the aspect of compatibility with values line, therefore after introducing anterior and posterior components, the star algorithm can be corresponding to the calculation results in (2), (3), (4) when calculation of a given fault tests is under its way.

After the introduction of both anterior and posterior components, the value fields of variable x is expanded, that is, x is not only taking its value from set $B_4 = \{\theta, I, D, \overline{D}\}$, but also taking its value from power set $P(B_4)$ of B_4. Fig. 1(b) has shown that the power set $P(B_4)$ consists in a 16-valued Boolean algebra for the logic operation of ".", "+" and "-", where elements $\{\theta\}$, $\{I\}$, $\{D\}$, $\{\overline{D}\}$ are its atoms.

From Theorem 1, variable x can be completely defined by its components when x takes its value on B_4, for those components are all orthognal, therefore, x can be expressed by those components' boolean sum when x takes its value on $P(B_4)$. Hence with x^0, x^1, x^*, $\overline{x}^*$ as its atoms, another 16-valued Boolean algebra isomorphic to $P(B_4)$ can be constructed (shown as Fig. 1(c)). In this way, by the transformation of Theorem 1, Lemma 1 and Eqs. (6), (7) and (8), the star algorithm expands the previous four-valued problem into 16-valued problem. We call this treatment the expansion transformation of Boolean algebra.

Definition 3. The star algorithm is an expansion transformation of Boolean algebra *: $B_4 \to B_2 4$, the transformation * is only defined by Theorem 1, Lemma 1 and Eqs. (6), (7) and (8).

By expansion transformation, the range of the variable's taking value is expanded, in practical application, the compatibility of variable's taking value can be considered. At the same time, four-valued Boolean operation becomes binary Boolean operation, which greatly simplify the calculation. In the generation of logic networks test, especially in the treatment of sequential circuit, there emerges wide application. It is easy to prove the following theorem.

Theorem 5. Under the transformation *: $B_4 \to B_2 4$, the operation properties of the element on B_4 are retained by Eqs.(6), (7) and (8).

Vector Expansion Transformation of General Boolean Algebra

Lemma 2. Let B_n be a non-degenerate Boolean algebra, there should exist a positive integer m, which makes the number of B_n's element be 2^m, namely $n = 2^m$ [6].

Let the element set of Boolean algebra B_n be $B_n = \{E_0 = \theta, E_1 = I, E_2, \ldots, E_{n-1}\}$, $B_n' = \{E_2, \ldots, E_{n-1}\}$, where θ is the minimal element and I, the maximal element. Suppose $N = \{0, 1, 2, \ldots, n-1\}$, $N' = \{2, 3, \ldots, n-1\}$.

Corollary 1. Suppose $|B_n| > 2$, then $\forall E_i \in B_n'$, there should exist a unique complementary element $E_j \in B_n'$, that is,

$$E_i + E_j = I, E_i \cdot E_j = \theta.$$

Theorem 6. Let $x \in B_n$. Taking value of every x in B_n, we choose variables x^0, x^1, $x^2, \ldots, x^{n-1}$ in B_2 corresponding to it, such that

$x = \theta$ iff $x^0 = 1$ and $x^i = 0$, $i \in N$, $i \neq 0$ ⎫

$x = I$ iff $x^1 = 1$ and $x^i = 0$, $i \in N$, $i \neq 1$ ⎬ (10)

$x = E_j$ iff $x^j = 1$ and $x^i = 0$, $i \in N$, $i \neq j$ ⎭

then

$$x = \sum_{i \in N} x^i E_i \qquad (11)$$

satisfies

$$\sum_{i \in N} x^i = 1 \qquad (12)$$

$$x^i \cdot x^j = 0, \quad i, j \in N, \quad i \neq j \qquad (13)$$

Eq.(11) is called standardized vector expression of x. And x is called vector and every x^i, $i \in N$ is called the components of x.

Theorem 7. Let

$$x = a_0 \cdot \theta + a_1 \cdot I + a_2 \cdot E_2 + \cdots + a_{n-1} E_{n-1}$$

be any vector expression of x, where $a_i \in B_2$, $i \in N$, then this vector expression can be turned into the standardized vector expression shown as Eq.(11), and

$$x^0 = \bar{a}_1 \cdot \bar{a}_2 \cdots \bar{a}_{n-1} = \bar{a}_1 \prod_{i \in N'} \bar{a}_i$$

$$x^1 = a_1 + \sum_{i,j \in N'} a_i a_j, \quad E_i, E_j \text{ complementary} \qquad (14)$$

$$x^i = \bar{a}_1 a_i \bar{a}_j, \quad i, j \in N', E_i E_j \text{ complementary}$$

Proof. First it is necessary to prove

$$\sum_{i \in N} x^i E_i = \sum_{i \in N} a_i E_i.$$

By the supposition we know $\displaystyle\sum_{i \in N} a_i = 1$.

If

$$a_0 \cdot (a_1 + a_2 + \cdots + a_{n-1}) \neq 0$$

then let

$$a_0 = a_0 \overline{(a_1 + a_2 + \cdots + a_{n-1})} + a_0 (a_1 + a_2 + \cdots + a_{n-1})$$

because $E_0 + E_i = E_i \ \forall \ i \in N$ hold, then follows

$$\sum_{i \in N} a_i E_i = a_0' \cdot E_0 + \sum_{i=1}^{n-1} a_i E_i,$$

where

$$a_0' = a_0 \overline{(a_1 + a_2 + \cdots + a_{n-1})}$$

and

$$a_0' + a_1 + a_2 + \cdots + a_{n-1} = 1,$$

$$a_0' \cdot (a_1 + a_2 + \cdots + a_{n-1}) = 0.$$

Let $x^0 = a_0'$, we get $x^0 = a_0' = \bar{a}_1 \cdot \bar{a}_2 \cdots \bar{a}_{n-1}$.

Notice: Here we have treated every a_i and every E_i as independent logic transformation.

In the following we should prove

$$a_1 \cdot I + \sum_{i,j \in N} a_i a_j I + \sum_{i,j \in N} \bar{a}_1 a_i \bar{a}_j E_i$$

$$= \sum_{i=1}^{n-1} a_i E_i, E_i, E_j \text{ complementary} \qquad (15)$$

Let $E_h \in B_n'$, by Corollary 1, there should be $E_k \in B_n'$, which makes $E_h = \bar{E}_k$ then from the first term of Eq.(15) we have

$$a_1 \cdot I = a_1 \cdot I + a_1 \cdot E_h + a_1 \cdot E_k.$$

Therefore factor $\bar{a}_1$ in the third term can be eliminated. And for any element a_h in the second term, there should be a_k, we have

$$a_h a_k \cdot I = a_h \cdot a_k \cdot E_h + a_h a_k \cdot E_k, \quad E_h = \bar{E}_k.$$

Simultaneously there should be the corresponding term in the third term

$$a_h \bar{a}_k E_h + \bar{a}_h a_k E_k. \text{(having eliminated factor } \bar{a}_1)$$

Adding up the two, we have

$$a_h \cdot a_k \cdot I + a_h \cdot \bar{a}_k \cdot E_h + \bar{a}_h a_k E_k = a_h E_h + a_k E_k.$$

Hence the first part proof is immediate. It is not difficult to prove

$$\sum_{i \in N} x^i = 1, \quad x^i \cdot x^j = 0, \quad i, j \in N, \quad i \neq j$$

that is, every $x^i (i \in N)$ satisfies the requirement of standardization. The proof is complete.

Theorem 8. Suppose $x, y \in B_n$, $x = \displaystyle\sum_{i \in N} x^i E_i$, $y = \displaystyle\sum_{i \in N} y^i E_i$ be the standardized vector expression, then for the operation $\cdot$, $+$, $-$, on B_n their components' recurrence formulas will be

$$(\bar{x})^i = x^j, \quad i, j \in N, \ E_i, E_j \text{ complemen-} \qquad (16)$$
$$\text{tary}$$

$$(x \cdot y)^0 = x^0 + y^0 +$$

$$\sum_{i,j \in N'} x^i \cdot y^j, \quad E_i, E_j \text{ complemen-} \qquad (17)$$
$$\text{tary}$$

$$(x \cdot y)^1 = x^1 y^1$$
$$(x \cdot y)^k = x^1 y^k + x^k y^1 + x^k y^k, \quad k \in N'$$
$$(x + y)^0 = x^0 y^0$$
$$(x + y)^1 = x^1 + y^1 + \sum_{i,j \in N'} x^i y^j, E_i, E_j \text{ com-} \qquad (18)$$
$$(x + y)^k = x^0 y^k + x^k y^0 + x^k y^k, k \in N' \text{ plementary}$$

Corollary 2. Suppose $F(x_1, \ldots, x_m)$ be any Boolean function of $B_n^m \to B_n$, then F can be given out by the vector expression, and its components can be recurrently calculated gate by gate according to Eqs. (16), (17) and (18) and the structural law of the function.

Let the power set of B_n be $\mathcal{P}(B_n)$, then we have

Lemma 3. Algebraic system $\mathcal{P}(B_n) = \langle \mathcal{P}(B_n),$

179

".", "+", "-", ϕ, B_n> is a 2^n-valued Boolean algebra.

By Theorem 6, variable x can be completely defined by its components x^0, $x^1,\ldots,x^{n-1}$ when it takes its value on B_n, since every component is orthogonal, then x can be expressed by each component's Boolean sum when it takes its value on $P(B_n)$. Let $W_n=\{x^0, x^1,\ldots,x^{n-1}\}$, $P(W_n)$ is the power set of W_n, we have

Lemma 4. Algebraic system $P(W_n)=\langle P(W_n)$, ".", "+", "-", 0, 1$\rangle$ is a Boolean algebra isomorphic to $P(B_n)$.

Theorem 9. There exists an expansion transformation of Boolean algebra as follows: $T:B_n \to B_2 n$, T is uniquely defined by Theorems 6, 7 and 8.

Corollary 3. Under transformation T, the operation properties of the element on B_n are retained by Eqs. (16), (17) and (18).

From the above results we could see that the vector expansion transformation $*$ of four-valued Boolean algebra is only a special case of transformation T. The essential of this transformation is through vector's form expanding the study of Boolean algebra B_n to the study of the power set that consists of B_n element, and the approach of study we adopted is the form of vector expression (i.e. the form of binary Boolean expression), the logic operation properties on B_n can be shown by these component expressions. Therefore, a new analytic method of Boolean algebra appears.

Example 1. Consider the vector transformation of $B_2=\{\theta,I,\cdot,+,-\} \to B_2^2$, there exist

$$x^0=1 \leftrightarrow x=\theta, \quad x^1=1 \leftrightarrow x=I, \quad x=x^0\cdot\theta +x^1\cdot I,$$
$$y=y^0\cdot\theta +y^1\cdot I, \quad (\bar{x})^0=x^1, (\bar{x})^1=x^0;$$
$$(x\cdot y)^0=x^0+y^0 \cdot (xy)^1=x^1 y^1;$$
$$(x+y)^0=x^0 y^0, (x+y)^1=x^1+y^1.$$

Virtually this is a general calculation problem of binary Boolean function.

Three-Valued Logic Vector Expansion Transformation

Consider the three-valued logic L_3 shown in Table 1, the relation diagram of which is shown as Fig. 2(a).

Theorem 10. Let logic variable $x \in L_3$, when x takes its value on L_3, we choose three proposition variables x^0, x^u, x^1 to be correspondent to it in B_2 such that

$$x= \theta \text{ iff } x^0=1 \text{ and } x^u=x^1=0,$$
$$x=I \text{ iff } x^1=1 \text{ and } x^0=x^u=0, \quad (19)$$
$$x=U \text{ iff } x^u=1 \text{ and } x =x^1=0,$$

then
$$x=x^0\cdot\theta +x^1\cdot I+x^u\cdot U \quad (20)$$
satisfies
$$x^0+x^u+x^1=1 \quad (21)$$
$$x^0\cdot x^u=x^u\cdot x^1=x^0\cdot x^1=0 \quad (22)$$

Eq.(20) is called the vector expression of x, and its components are x^0, x^1, x^u.

Theorem 11. Let $x=a\cdot\theta+b\cdot I+c\cdot U$ be any vector expression, then the following formulas can be turned into the standardized vector expression as Eq.(20).
$$x^0=\overline{bc}, x^1=c, x^u=\bar{b}c \quad (23)$$

Theorem 12. Let
$$x,y \in L_3,$$
$$x=x^0\cdot\theta+x^1\cdot I+x^u\cdot U,$$
$$y=y^0\cdot\theta +y^1\cdot I+y^u\cdot U$$

be standardized vector expression, the basic component calculation formulas of operations $-, \cdot, +$ of x and y on L_3 are
$$(\bar{x})^0=x^1, \quad (\bar{x})^1=x^0, \quad (\bar{x})^u=x^u, \quad (24)$$
$$(x\cdot y)^0=x^0+y^0, \quad (x\cdot y)^1=x^1 y^1,$$
$$(x\cdot y)^u=x^1\cdot y^u+x^u y^1+x^u y^u, \quad (25)$$
$$(x+y)^0=x^0 y^0, \quad (x+y)^1=x^1+y^1,$$
$$(x+y)^u=x^0 y^u+x^u y^0+x^u y^u, \quad (26)$$

Due to the expansion produced by the transformation, it is possible for us to handle the compatibility on L_3 when the variable takes its value. In Fig. 2(c) $\bar{x}^0$, $\bar{x}^u$, $\bar{x}^1$ show us the compatible characteristics of taking value, where
$$\bar{x}^0=x^u+x^1, \quad \bar{x}^u=x^0+x^1, \quad \bar{x}^1=x^0+x^u.$$

It is easy to prove the correctness of the following formulas:
$$(x\cdot y)^u=\bar{x}^0 y^u+x^u \bar{y}^0, \quad (25)'$$
$$(x+y)^u=\bar{x}^1 y^u+x^u \bar{y}^1, \quad (26)'$$
$$(x\oplus y)^0 x^1 y^1+x^0 y^0,$$
$$(x\oplus y)^1=x^1 y^0+x^0 y^1, \quad (27)$$
$$(x\oplus y)^u=x^u+y^u,$$
$$(\bar{x})^0=\overline{x^1}, \quad (\bar{x})^1=\overline{x^0}, \quad (\bar{x})^u=\overline{x^u} \quad (28)$$
$$(x\cdot y)^0=\overline{x^0}\,\overline{y^0}, \quad (x\cdot y)^1=\overline{x^1}+\overline{y^1}$$
$$\overline{(x\cdot y)}^u=x^1 y^1+x^0_+ y^0=x^1 y^{\bar{u}}+x^0+y^0= \quad (29)$$
$$x^u y^1+x^0+y^0=x^u y^u+x^0+y^0$$
$$\overline{(x+y)}^0=\overline{x^0}+\overline{y^0}, \quad \overline{(x+y)}^1=\overline{x^1}\,\overline{y^1}$$
$$\overline{(x+y)}^u=x^0 y^0+x^1+y^1=x^0 y^{\bar{u}}+x^1+y^1= \quad (30)$$
$$x^u y^0+x^1+y^1=\overline{x^u}\,\overline{y^u}+x^1+y^1.$$

Example 2. consider the reset problem of synchronous R-S flip-flop [15]. When R=S=1, under the action of synchronous pulse at end A, R-S flip-flop will be reset. But by the direct use of three-valued logic simulation we cannot make the decision. Now let us analyse this by means of

our method. In order to be simple, we assumed the delay only occurs outside the feedline.

$$T_i^0 = (C_i + F_i)^1 = C_i^1 + F_i^1 = A_i^1 R_i^1 + D_i^0 T_{i-1}^0$$

$$= A_i^1 B_i^1 T_{i-1}^1 + (A_i^0 + S_i^0) T_i^0{}_{-1}$$

$$= A_i^1 B_i^1 S_i^0 (T_{i-1}^1 + T_{i-1}^0) + A_i^1 B_i^1 T_{i-1}^1 + A_i^0 T_{i-1}^0$$
$$+ S_i^0 T_{i-1}^0$$

$$= A_i^1 B_i^1 S_i^0 \overline{T_{i-1}^u} + A_i^1 B_i^1 T_{i-1}^1 + A_i^0 T_{i-1}^0 + S_i^0 T_{i-1}^0,$$

where the first term $A_i^1 B_i^1 S_i^0 T_{i-1}^u$ shows that when $A=B=\overline{S}=1$, no matter what the previous state of R-S flip-flop is (actually it only takes the definite state values 0,1), it can be synchronized to the state of T=0. Hence we see that T^u and T^u is different in meaning. This means a new simulation method can be adopted, i.e. add up another state 0/1, or handle the simulation problem by means of component $\overline{x}^u$.
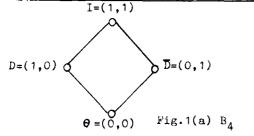
Conclusion
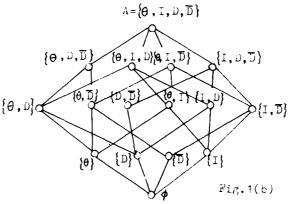
We have discussed the mathematical structure of the star algorithm and generalized the vector expression into the general logic system. It is worthwhile to point out that this expression is available to any logic algebraic system which defined operation ·, +, -. It is only when the operation in these systems satisfy the bidirectional distribution law, de Morgan law that we can use the component formula to calculate recurrently.

Because the compatibility can be dealt with the method developed, it brings the simplication of the calculation on one hand and on the other the function of describing the problem is also strengthened. In addition, due to the strictness of our method, the recurrence formula established could completely show out the rule of logic operation. It is easy to make comprehensive consideration to the problem, therefore, it can be used to test the correctness of a new calculation process. Finally, by the use of the vector transformation we can deal with those general multiple-valued logic problems. Therefore, its prospect of application is promising.

∗ ∗ ∗   ∗ ∗ ∗   ∗ ∗ ∗   ∗ ∗ ∗
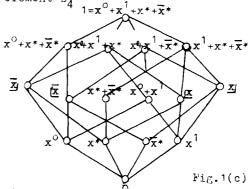
Fig.1. Four-valued Boolean algebra

B_4 and its expansion transformation

I=(1,1)

D=(1,0) $\overline{D}$=(0,1)

θ=(0,0)   Fig.1(a) B_4

$A=\{θ, I, D, \overline{D}\}$

Fig.1(b)

power set algebra constructed by element B_4
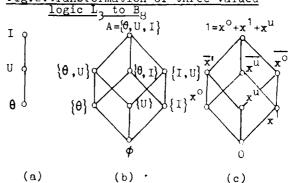
$1 = x^0 + x^1 + x^* + \overline{x}^*$

Fig.1(c)

16-valued Boolean algebra B_{16} constructed by the components of variable x in the vector expression of B_4

Table 1. Three-valued logic operation rule

·	θ	I	U
θ	θ	θ	θ
I	θ	I	U
U	θ	U	U

(a)

+	θ	I	U
θ	θ	I	U
I	I	I	I
U	U	U	U

(b)

x	$\overline{x}$
θ	I
I	θ
U	U

(c)
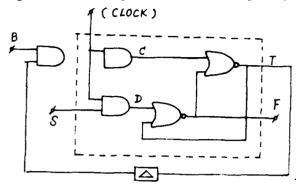
Fig.2. Transformation of three-valued logic L_3 to B_8

I
U
θ

$A=\{θ, U, I\}$

$1 = x^0 + x^1 + x^u$

(a)   (b) ·   (c)

(a) relation diagram of L_3

(b) power set algebra constructed by
$A=$, I, U

(c) 8-valued Boolean algebra B_8 constructed by the components of variable x in the vector expression of L_3

Fig.3. Reset of synchronous R-S flip-flop

References

1 Chen Tinghuai, Four-valued logic and star algorithm, Chinese Journal of Computer, Vol. 2, No. 4, 10(1979), 243-264.

2 S. B. Akers, Jr., A logic system for fault test generation, IEEE Trans. Comput., Vol. C-25(1976), 620-630.

3 P. Muth, A nine-valued circuit model for test generation, IEEE Trans. Comput., Vol. C-25(1976), 630-636.

4 C. W. Cha, et al, Nine-valued algorithm for test pattern generation of combinational digital circuits, IEEE Trans. Comput., Vol. C-27(1978), 193-200.

5 M. A. Breuer, A note on three-valued logic simulation, IEEE Trans. Comput., Vol. C-21(1972), 399-402.

6 Yuan Youguang, Chen Tinghuai, The dynamic testing of combinational logic networks, FTCS-12, Santa Monica, 1982, 173-180.

ROOTS OF N - VALUED SWITCHING FUNCTIONS[+]

Corina Reischer* and Dan A. Simovici**

*Departement de Mathematiques
Universite due Quebec a Trois-Rivieres
Trois-Rivieres C.I.500, Que, Canada G9A 5H7

**Department of Mathematics
University of Massachusetts at Boston.
Harbor Campus, Boston, MA 02125, USA

Abstract

A root of order u of a transformation f of a finite set is a transformation of this set which iterated for u times gives f. The existence of such roots is discussed. The paper offers a new insight on the existence of square roots of Boolean functions as well as an algorithm for extracting the roots of q - ary, p - valued switching functions.

I. Introduction

In a previous paper [3] we have established certain iteration properties of multivalued switching functions. Let <p> be the set $\{0, \ldots, p-1\}$. It was proved [3] that for any p - valued, q - ary switching function F : $<p>^q \rightarrow <p>^q$ we have $F^k = F^\ell$ iff $\ell \geq p^q - 1$ and $k \equiv \ell \pmod{\kappa(p^q)}$, where $\kappa(m)$ is the least common multiple of the numbers $1, \ldots, m$.

The aim of this paper is to study the existence of u - ary roots of p - valued switching functions.

Let $f : <m> \rightarrow <m>$ be a transformation of the set <m>.

<u>Definition 1.</u> A function $\alpha : <m> \rightarrow <m>$ is a <u>u - ary root</u> of the function $f : <m> \rightarrow <m>$ if $\alpha^u = f$. The powers are considered here with respect to the iteration of functions.

If $F : <p>^q \rightarrow <p>^q$ is a q - ary, p - valued switching function we shall encode it as a function $f : <m> \rightarrow <m>$ (see the IVth section) and reduce the existence of roots of such functions

to the previous problem.
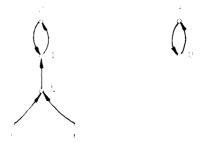
We shall recapture and explain from a new point of view our previous results from [4], regarding the existence of square roots of binary (Boolean) switching functions. In view of the aforementioned result of [3] it makes sense to study roots of order up to $p^q - 2 + \kappa(p^q)$ for a function F : $<p> \rightarrow <p>^q$.

II. Simple Functions and the Spectrum of a Function

Let $f : <m> \rightarrow <m>$ be a function and let us consider the directed graph $G_f = (<m>, E_f)$, having <m> as set of vertices; the set of edges is $E_f = \{(x,f(x)) \mid x \in <m>\}$. Since the out - degree of each vertex $x \in <m>$ is 1 it is clear that G_f consists of oriented cycles to which trees may be attached by their roots. When f is a permutation, the graph G_f consists only from cycles.
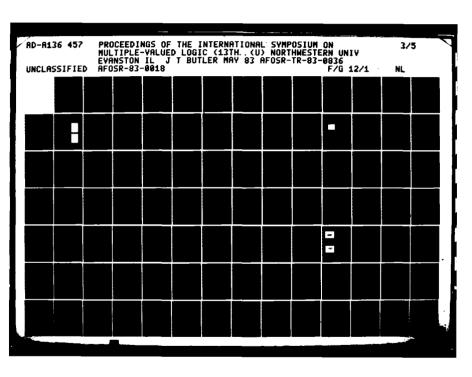
For instance, the graph of the function $f : <7> \rightarrow <7>$, given by
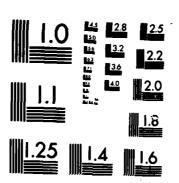
x	0	1	2	3	4	5	6
f x	2	2	1	3	1	4	4

is represented in Fig. 1.

Fig. 1

MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

Definition 2. A <u>simple function</u> is a function
$h : \langle m \rangle \rightarrow \langle m \rangle$, $h \neq 1_m$, whose graph G_h contains an
cycle C such that every vertex $x \in \langle m \rangle$ satisfies
one of the following two conditions:

    i) either x is a fixed point of h (i.e.
       $h(x) = x$) or

    ii) there exists $b \in N$ such that $h^b(x) \in C$.

We have denoted by 1_m the identity mapping on
the set $\langle m \rangle$.

Theorem 1. For every function $f : \langle m \rangle \rightarrow \langle m \rangle$,
$f \neq 1_m$, there exists a family of simple functions
$\{h_1, \ldots, h_\ell\}$ such that $f = h_1 \circ \ldots \circ h_\ell$ and
$h_i \circ h_j = h_j \circ h_i$, for $1 \leq i, j \leq \ell$. We shall re-
fer the functions $h_1, \ldots, h_\ell$, as the <u>components</u> of
the function f.

    Proof Let $\{C_1, \ldots, C_\ell\}$ be the set of cycles
of the graph G_f and let D_i be the set of vertices
belonging to the cycle C_i or to one of the trees
attached to it, for $1 \leq i \leq \ell$. Clearly, $D_i \cap D_j = \emptyset$, for $1 \leq i, j \leq \ell$, $i \neq j$. We remark also that
$x \in D_i$ implies $f(x) \in D_i$.

    The simple functions $\{h_i \mid 1 \leq i \leq \ell\}$ are
given by

$$h_i(x) = \begin{cases} f(x), \text{ if } x \in D_i, \\ \\ x, \text{ otherwise,} \end{cases}$$

for $1 \leq i \leq \ell$.

    Based on the previous remarks, the reader can
easily verify that $h_i \circ h_j = h_j \circ h_i$. Moreover,
if $x \in D_i$, we have
$(h_1 \circ \ldots \circ h_{i-1} \circ h_i \circ h_{i+1} \circ \ldots \circ h_\ell)(x) =$

$= (h_1 \circ \ldots \circ h_{i-1} \circ h_i)(x)$

$= (h_1 \circ \ldots \circ h_{i-1})(h_i(x))$

$= f(x)$.   *******

Definition 2. The <u>spectrum</u> of a function $f :$
$\langle m \rangle \rightarrow \langle m \rangle$, $f \neq 1_m$, is a sequence of natural numbers.

$$\text{Spec}(f) = (s_1, s_2, \ldots, s_n, \ldots) ,$$

where s_n is the number of simple functions, compo-
nents of f, whose unique cycle has length n;
$\text{Spec}(1_m) = (m, 0, 0, \ldots)$.

    Clearly, since f is a transformation of a fi-
nite set, only a finite number of initial components
of Spec(f) are non - null. If v =
$(v_1, \ldots, v_\ell, \ldots)$ is a sequence of numbers for
which $\sum \ell v_\ell \leq m$ there exist at least

$$\frac{m!}{(1!)^{v_1} \ldots (\ell!)^{v_\ell} \ldots}$$

transformations of the set $\langle m \rangle$ having the spectrum
v.

Example 1. For the transformation f whose graph is
given in Fig. 1, we have $\text{Spec}(f) = (0, 2, 0, \ldots)$.
The graphs of the components of f are given in
Fig. 2a and Fig. 2b.
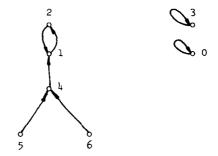
Fig. 2a
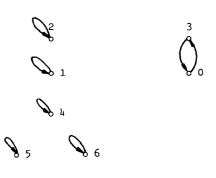
Fig. 2b

III.   <u>Roots of Permuations</u>

    If $\{h_1, \ldots, h_\ell\}$ is the set of components of
the function $f : \langle m \rangle \rightarrow \langle m \rangle$, then for every
$u \in N$, $f^u = h_1^u \circ \ldots \circ h_\ell^u$. However, we remark
that if $h : \langle m \rangle \rightarrow \langle m \rangle$ is a simple function whose
graph G_h contains a cycle of length i then its
iteration h^u is not, in general, a simple function
because G_{h^u} will contain (i,u) cycles of length
$i/(i,u)$. Here (i,u) stands for the greatest common
divisor of i and u.

Let $A_{u,\ell}$ be the set

$$A_{u,\ell} = \{i \mid i \in N, \; i/(i,u) = \ell\} \;,$$

for $u, \ell \in N$. Assume that

$$u = p_1^{\beta_1} \cdots p_k^{\beta_k} \quad \text{and} \quad \ell = p_1^{\alpha_1} \cdots p_k^{\alpha_k} \;,$$

where $p_1, \ldots, p_n, \ldots$ is the sequence of primes.

If $i = p_1^{\gamma_1} \cdots p_k^{\gamma_k}$ then we have

$$\gamma_j - \min(\beta_j, \gamma_j) = \alpha_j, \quad \text{for } 1 \le j \le k \;.$$

We obtain two cases:

i) $\alpha_j = 0$, which allows γ_j to run from 0 to β_j and

ii) $\alpha_j \ne 0$; in this case $\gamma_j = \alpha_j + \beta_j$.

Therefore, $A_{u,\ell}$ contains

$$\prod \{(1 + \beta_j) \mid 1 \le j \le k, \; \alpha_j = 0\}$$

elements.

Let us consider the transformations f, g: $\langle m \rangle \to \langle m \rangle$, such that $f = g^u$, $\text{Spec}(f) = (t_1, \ldots, t_\ell, \ldots)$ and $\text{Spec}(g) = (s_1, \ldots, s_n, \ldots)$. In view of the above discussion, a cycle of length ℓ of G_f can arise from splitting a cycle of length i of G_g into (i,u) cycles $i/(i,u) = \ell$. Therefore,

$$t_\ell = \sum \{(i,u)s_i \mid i \in A_{u,\ell}\} \;, \tag{1.ℓ}$$

for $\ell \ge 1$. We have proved that the existence of a root of order u of f implies the solvability of the system (1.ℓ) in natural numbers, for $s_1, \ldots, s_n, \ldots$

Theorem 2. Let $f : \langle m \rangle \to \langle m \rangle$ be a permutation of $\langle m \rangle$ and let u be a prime number. This function has a root of order u iff $\ell \equiv 0 (\bmod\ u)$ implies $t_\ell \equiv 0 (\bmod\ u)$, where $\text{Spec}(f) = (t_1, \ldots, t_\ell, \ldots)$.

Proof Since u is prime, $u = p_d$, we have $\beta_d = 1$ and $\beta_j = 0$, for $1 \le j \le k$, $j \ne d$. Therefore, $A_{u,\ell} = \{\ell, \ell u\}$ if $\alpha_d = 0$(i.e. if ℓ is not a multiple of $u = p_d$); otherwise, $A_{u,\ell} = \{\ell u\}$. The system (1.ℓ) can be rewritten as

$$t_\ell = s_\ell + us_{\ell u}, \qquad \text{if } \ell \not\equiv 0 \;(\bmod\ u)$$

$$t_\ell = us_{\ell u}, \qquad \text{if } \ell \equiv 0 \;(\bmod\ u)$$

and this clearly implies that $t_\ell \equiv 0(\bmod\ u)$ if $\ell \equiv 0 \;(\bmod\ u)$, which proves the necessity of the condition of the Theorem.

It is clear that the previous condition is sufficient because it implies that (1.ℓ) is consistent. Moreover, under the conditions of the Theorem, the system (1.ℓ) has $\prod\{\lceil t_\ell/u\rceil \mid \ell \not\equiv 0 \;(\bmod\ u)\}$ solutions.***

Example 2. Let us condsider the permutation $f : \langle 7 \rangle \to \langle 7 \rangle$ defined by the following Table

x	0	1	2	3	4	5	6
f(x)	5	2	1	4	3	6	0

whose graph is given in Fig. 3.

Fig. 3
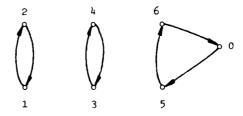
The spectrum of f is $\text{Spec}(f) = (0, 2, 1, 0, \ldots)$, hence it is clear that it is possible to extract the square root of the 7 - valued function. The spectrum of the root can be obtained from the following system:

$$0 = s_1 + 2s_2$$

$$2 = 2s_4$$

$$1 = s_3 + 2s_6 \;.$$

Therefore, the spectrum of the root can be only $(0, 0, 1, 1, 0, \ldots)$ and there are at least $7!/(3!4!) = 35$ functions having this spectrum.

To find a root we have to join the two cycles of length 2 into a cycle of length four; the cycle of length three comes from a cycle having the same length. The graph of one of the roots is given in Fig. 4

Fig. 4

185
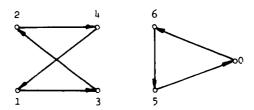
In general, we can find the roots of order u of a permutation $f : <m> \to <m>$, where u is a prime, by applying the following algorithm:

i) Verify if Spec(f) satisfies the conditions from Theorem 2.

If yes, proceed to step ii); otherwise stop.

ii) Determine those components s_n of the spectrum of the root for which $n \equiv 0 \pmod{u^2}$ by $s_n = t_{n/u}/u$. If $n \not\equiv 0 \pmod{u^2}$ but $n \equiv 0 \pmod u$ choose $s_n \leq \lceil t_{n/u}/u \rceil$; $s_{n/u}$ is given in this case by $s_{n/u} = t_{n/u} - us_n$.

iii) After choosing the spectrum of the root among different possible spectra, assuming that $\ell \not\equiv 0 \pmod u$ and $t_\ell = s_\ell + us_{\ell u}$ we shall "aggregate" u cycles of length ℓ of the graph G_f into a cycle of length ℓu by using the following procedure. If these cycles are $C_1, \ldots, C_u$ and $x_1, \ldots, x_u$ are arbitrary vertices on these cycles, respectively, we shall consider the edges $(x_1,x_2), (x_2,x_3), \ldots, (x_{u-1},x_u)$. The vertex x_u will be joined to vertex x_1' following x_1 on the cycle C_1 ; x_1' will be joined with x_2', the successor of x_2 on C_2, etc.

The number of distinct cycles of length ℓu, which can be obtained from a set of u cycles of length ℓ , is dependent on the number of ways in which we join the cycles of length ℓ (modulo circular permutations of selected joining orders) and on the number of choices of the initial vertex in every cycle of length ℓ (modulo these choices determined by the initial choice). Thus, we obtain $(u-1)! \ell^{u-1}$ modalities of generating a cycle of length ℓu from a set of u cycles length ℓ ; there are $(u-1)! \ell^{u-1} \binom{us_{\ell u}}{u}$ distinct cycles of length ℓu.

Example 3. Let $f : <32> \to <32>$ be the permutation whose graph is given in Fig. 5. Its spectrum is Spec(f) = (0, 7, 6, 0, 0, ...) .
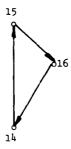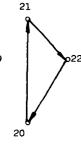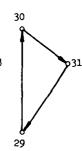
Fig. 5

The equations (1.ℓ) give
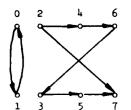
$$t_2 = s_2 + 3s_6$$
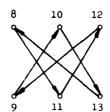
$$t_3 = 3s_9$$

By choosing $s_6 = 2$ we obtain the following spectrum for the cubic root :

Spec(g) = (0, 1, 0, 0, 0, 2, 0, 0, 2, 0, ...)

and the following graph:

Fig. 6

where $X = (x_1, .. \ x_i, ..., x_n)$ and $X' = (x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$ and let us fix the $(n-1)$ – tuple X'. According to the values assumed by h and k we can have four different aspects of the function $\psi_{X'} : <2> \to <2>$, where $\psi_{X'}(x_i) = x_i \ h(X') + \bar{x}_i \ k(X')$ (see Fig. 7).

$\psi_{X'}(x_i) = 0$ $\psi_{X'}(x_i) = 1$

$Spec(\psi_{X'})=(1,0,...)$ $Spec(\psi_{X'})=(1,0,...)$

$\psi_{X'}(x_i) = x_i$ $\psi_{X'}(x_i) = \bar{x}_i$

$Spec(\psi_{X'})=(2,0,0,...)$ $Spec(\psi_{X'})=(0,1,0,...)$

Fig. 7

It is evident that no square root exists in the case when $\psi_{X'}(x_i) = \bar{x}_i$, i.e. when $\psi_{X'}$ is negative in x_i. We retrieved the necessity of the positivity of f in x_i established in [2].

Let now $F : <p>^q \to <p>^q$ be a q – ary, p – valued switching function. We can apply our previous arguments to the function F by encoding each q – tuple $(a_1, ..., a_q)$ as a number in the basis p, via the bijection $\phi : <p>^q \to <p^q>$. The function $f: <p^q> \to <p^q>$ is given by $f(n) = \phi(F(\phi^{-1}(n)))$. If there exists a root g of order u of f then the function $G(a_1, ..., a_q) = \phi^{-1}(g(\phi(a_1, ..., a_q)))$, for $(a_1, ..., a_q) \in <p>^q$ is a root of the function F and every such root can be obtained by the above mechanism.

Eample 4. Let us consider the 3 – valued, 2 – ary function $F : <3>^2 \to <3>^2$ given by Table 1.

It is clear that if there exists a number $\ell \equiv 0 \ (\text{mod } u)$ such that the ℓ – th component of the spectrum of the function $f : <m> \to <m>$ is not a multiple of u (i.e. $t_\ell \not\equiv 0 \ (\text{mod } u)$ then f does not have any root of order u.
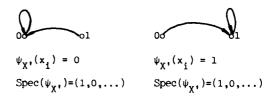
For instance, there is no square root of the function considered in Example 3, because $t_2 = 7 \not\equiv 0 \ (\text{mod } 2)$.

IV. Miscellaneous Aspects of Extracting Roots

Let $f, g : <2>^n \to <2>$ be two Boolean functions of n variables. In our previous paper [2] we have considered the special product "o_i", where

$f \ o_i \ g(x_1, ..., x_n) = f(x_1, ..., x_{i-1}, g(x_1, ..., x_n), x_{i+1}, ..., x_n)$. By regarding $x_1, ..., x_{i-1}, x_{i+1}, ..., x_n$ as parameters this operation is essentially a superposition of one variable function.

Assume that the Shannon's decomposition of f is
$$f(X) = x_i \ h(X') + \bar{x}_i k(X') ,$$

		Table 1	
x_1	x_2	$F_1(x_1,x_2)$	$F_2(x_1,x_2)$
0	0	0	1
0	1	1	2
0	2	1	0
1	0	2	1
1	1	0	0
1	2	1	1
2	0	0	2
2	1	2	0
2	2	2	2

		Table 2	
x_1	x_2	$G_1(x_1,x_2)$	$G_2(x_1,x_2)$
0	0	0	2
0	1	1	0
0	2	0	1
1	0	1	2
1	1	2	0
1	2	2	1
2	0	0	0
2	1	1	1
2	2	2	2

Table 1'

m	f(m)
0	1
1	5
2	3
3	7
4	0
5	4
6	2
7	6
8	8

Table 1' contains the encoded version of Table 1; the graph of f is given in Fig 8.

Fig. 8

References

[1]. M. Davio, J.-P. Deschamps, A Thayse, "Discrete and Switching Functions", McGraw-Hill International Book Company New York, 1978.

[2]. C. Reischer, D. A. Simovici, "Associative Algebraic Structures in the Set of Boolean Functions and some applications in Automata Theory", IEEE Trans. Computers, vol C-20, (1971), pp. 298-303.

[3]. C. Reischer, D. A. Simovici, "Several Remarks on Iteration Properties of Switching Functions", Proceedings of the Twelfth International Symposium on Multiple-Valued Logic, May 1982, Paris, France, pp. 244-247.

[4]. S. Rudeanu, "Square Roots and Functional Decomposition of Boolean Functions", IEEE Trans. Computers, vol. C-25, (1976), pp. 528-532.

and Spec(f) = (1, 0, 0, 2, 0, ...). It is clear that it is possible to extract a square root from f ; The Tables 2' and 2 contain the definition of one of the square roots in its encoded and decoded form, respectively.

Table 2'

m	g(m)
0	2
1	3
2	1
3	5
4	6
5	7
6	0
7	4
8	8

Session 5A
Circuit and Technology II

A QUATERNARY LOGIC ENCODER-DECODER CIRCUIT DESIGN USING CMOS*

David A. Freitas and K. Wayne Current
Integrated Circuits Laboratory
Department of Electrical and Computer Engineering
University of California, Davis
Davis, California 95616

ABSTRACT

A binary-to-quaternary encoder and quaternary-to-binary decoder circuit pair is described as designed in a 5-volt CMOS technology. These circuits communicate with logical currents. Using model parameter values for a standard 5-micron polysilicon gate process technology and 10 microamp logical currents, we have simulated propagation delays of about 20 ns from binary encoder input to binary decoder output. With the encoder using scaled-up logical currents and driving a 100 pF load on the decoder input to simulate communication between chips, we observe simulated worst-case delays of about 35ns.

INTRODUCTION

Much of the recent research in multiple-valued logic (MVL) circuits has dealt with performing arithmetic and logical operations with quaternary logical signals on-chip where the noise environment is considered less severe. Advantages claimed are reduced signal interconnect lines, fewer devices, and reduced chip area at the expense of increased delay times. The advantage of reduced signal lines may also be important in easing some of the packaging pin-out limitations of LSI and VLSI circuits. Rather than multiplexing signals on one pin, we can encode two digits of binary data simultaneously on one pin as a quaternary logical signal for delivery off-chip. This quaternary signal is decoded in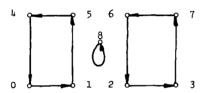to its two binary digits by the input stage of the receiving chip. Depending upon the application, the delay penalty for this method of data transmission may be easier to tolerate than the additional complexity of multiplexing circuitry. However, the quaternary logical system we might define must allow for reduced noise margins when we put twice as many logical voltage levels in the same total voltage swing available for binary signals. Transmission and decoding of quaternary-valued logical currents will not be directly limited by the power supply voltage. Traditional noise margin concepts will still apply when the currents are converted to voltages, but we may be able to take advantage of the analog summing of logical currents at a node to provide improved performance or additional logical functions before conversion to voltage. Current outputs have the

disadvantage that they cannot be bussed to multiple loads. However, in some digital signal processing and interfacing applications single loads are expected and in these applications current outputs should function well.

The trend in LSI and VLSI circuit design and process development seems to be toward two technologies, ECL and silicon-gate CMOS, that will dominate new chip developments. ECL speeds and chip densities are improving each year. Only GaAs can seriously challenge the overall performance of ECL, but GaAs gate densities are still very low. Silicon-gate CMOS processes are being developed and refined to the point that CMOS LSI circuits can in general out-perform most TTL-based circuits. Gate counts and chip sizes of VLSI CMOS far exceed those of VLSI LSTTL chips. Thus, we see ECL and silicon-gate CMOS dominating VLSI chip designs in the near future. For that reason, we have turned our attention to some MVL circuits for realization in CMOS.

Over the past 10 years, a substantial amount of work has been published on MOS MVL circuits; see, for example, [1-23]. In this paper, we describe the design and operation of a CMOS binary voltage-to-quaternary current encoder circuit and a CMOS quaternary current-to-binary voltage decoder circuit that are input-output compatible. These current-mode circuits are designed for use in a standard 5-volt CMOS technology and all binary voltages are completely compatible with standard CMOS logic. Thus, the encoder-decoder combination described is completely realizable in present CMOS processes.

CIRCUIT DESCRIPTION AND OPERATION

The encoder-decoder circuit combination to be described is designed to serve both on-chip and off-chip interface functions. With proper scaling of device areas the encoder circuit can drive larger capacitive loads with reduced propagation delays and the decoder can maintain its high degree of logical discrimination.

Any two binary signals can be assumed to represent a binary weighted number. That two-digit number can then be encoded into a single-digit base-four equivalent number. Thus, it is well known that we can reduce the number of signal lines required to transfer this information from one location to another to half that required for

190

binary signals. With fewer signal lines containing more information, some of the difficult pin-out limiting problems in LSI design may be alleviated. In an effort to provide this quaternary-to-binary encoding and binary-to-quaternary decoding, we have examined several schemes using logical voltage signals and logical current signals in enhancement NMOS, enhancement-depletion NMOS, and CMOS technologies. The encoder circuit shown in Figure 1 and the decoder circuit shown in Figure 2 are one realization of these functions that we have developed for CMOS.

A simplified schematic of the encoder circuit is shown in Figure 1. A reference current is established and duplicated by transistors T1-T4. Notice that the current in T4 is twice as large as that in T3. The binary signals to be encoded are input to the pass transistors T5 and T6, where the signal assigned the most significance is applied to the gate of T6 which will pass the doubly weighted current. The pass transistor sources are tied together to form the analog sum of the currents. The binary inputs are derived from CMOS logic gates that swing the full power supply voltage, 5 volts. The summed source currents, I_0, is the four-valued encoded output current.

The encoder's four-valued output current is connected either on-chip or off-chip to the compatible current comparator section of the decoder circuit shown in a simplified schematic in Figure 2a. The four-valued current is applied to the drain of the decoder's input transistor T7. T7 develops a gate-to-source voltage that then drives three current comparators made up of transistor pairs T8-T9, T10-T11, and T12-T13. The common drain connection of each current comparator transistor pair is labeled A, B, and C, respectively. Voltages A, B, and C will remain HIGH as long as the input current is less than one-half the logical output current increment, I. For an input current greater than .5I, A will go LOW, while voltages B and C remain HIGH. For an input current greater than 1.5I, B will also go LOW and C will remain HIGH. Input currents greater than 2.5I will drive C to the LOW state and all three comparators will be LOW. The three CMOS-compatible logical voltages A, B, and C then drive the standard CMOS decoding logic gates shown in Figure 2b. The decoding logic recreates the two binary logical voltages in the same order of significance that they were applied.

In the next section, we examine in more detail the performance of the current comparators. Input-output characteristics of the encoder-decoder combination are also presented and evaluated in the sections that follow.

EVALUATION OF THE CURRENT COMPARATORS

The operation of the current comparators [24] must be examined to evaluate the decoder's ability to discriminate between the different input current levels. Since the three current comparators differ only in their threshold currents, we will analyze the operation of the C output comparator and then list the specifics of each comparator used in the

decoder circuit. The three current comparators used in the decoder exhibit the DC input-output transfer characteristic shown in Figure 3. Referring to Figure 2a, the decoder input current, I_{IN}, is applied to the drain of the diode-connected N-channel transistor T7 where it generates V_{GS_7}. Since we want the voltage C to fall to a LOW value when its comparator threshold current is exceeded, transistor T12 will be operating in the linear region. For T7 in the saturation region,

$$I_{IN} = K_7 (W/L)_7 (V_{GS_7} - V_{TH_7})^2 ,$$

and for T12 in the linear region,

$$I_{D_{12}} = K_{12} (W/L)_{12} (2(V_{GS_{12}} - V_{T_{12}})V_{DS_{12}} - V_{DS_{12}}^2).$$

The identical NMOS transistors (with identical width-to-length ratios) are assumed to have identical K's, and threshold voltages. For ease of this discussion we will use K=1E-5 and a 1 volt threshold. We must force $V_{DS_{12}}$ to be low enough to turn off the gate that it will be driving. For this calculation, we set the maximum allowable LOW logic level of C to be about one-third the NMOS threshold voltage, or about (1/3) volt for this discussion. All that remains to be specified is the relationship between the input current and the threshold current. Let us assume for now that the LOW C output occurs when the I_{IN} current is 1.1 times as large as the I_{TH_C} current. Thus, when C is LOW,

$$I_{IN} = 1.1 I_{TH_C} .$$

Using the two drain current equations and the relationship between drain currents we have selected, we can solve for the V_{GS} required to satisfy all these relationships. A value $V_{GS} \cong 1.5$ volts results. Given this V_{GS} value and the value of the input current to be detected, we can solve for the width-to-length ratio of the transistors T7 and T12. Transistors T14 and T15 in Figure 2a establish a reference current, 2I, that is mirrored and amplified by factors .25, .75, and 1.25 by transistors T9, T11, and T13, respectively, to establish the three threshold currents. For the A logical output, the threshold current I_{TH_A} is .5I. For B and C outputs, the threshold currents are $I_{TH_B} = 1.5I$, and $I_{TH_C} = 2.5I$, respectively. In the simplified encoder-decoder shown in the figures, we are using logical levels of 0, 10, 20, and 30 microamps. Current comparator C is to provide a

HIGH output voltage for input currents less than 25 microamps and a logical LOW voltage for input currents greater than 25 microamps. Thus, the threshold current for current comparator C is 25 microamps. For the data used in this example, width-to-length ratio of 10 is required to provide the necessary voltage swing.

Greatest comparator discrimination is obtained by using maximum comparator gain. This current comparator configuration converts the input current to a voltage, V_{GS}, that drives a common source amplifier with active, current source, loading. Another way to describe the operation of this circuit is to consider it a current mirror that reproduces I_{IN} as I_D, and I_D drives a high-impedance active, current source, load to convert the current difference to a voltage. Either way, we can analyze the comparator and find the transresistance amplifier gain, R_0, to be the parallel combination of the output resistances of the n-channel driver and p-channel load devices;

$$R_0 \ (V_c/I_{IN}) = r_{o_p} || r_{o_n} = (I_{D_p} \lambda_p)^{-1} || (I_{D_n} \lambda_n)^{-1} ,$$

$$R_0 = (I_D (\lambda_p + \lambda_n))^{-1} ,$$

for the active load connection, where λ represents the channel length modulation effects and has units of V^{-1}. A large gain is desired to provide a sharp comparator transition and greater noise margin. Lower I_{TH} values will increase the gain at the expense of greater comparator delay times. One can increase the interface driving current, I_{IN}, by, for example, an order of magnitude to provide increased capacitive loading drive capability independently of the threshold currents and still maintain the same comparator current levels and gain by appropriately designing the width-to-length ratios of transistors T7 and transistors T8, T10, and T12. The comparators and decoder performance will be unchanged. For example, using $10I_{IN}$ instead of I_{IN} for interfacing, we will need to increase the width of T7 by a factor of 10. This feature allows considerable design flexibility. We could apply the same technique to each comparator to give them all the same low quantity of drain current and, thus, the same high value of gain and still detect the same three input current levels selected previously. The trade-off here is the reduced load driving current available in the scaled-down current comparators. These scaled-down threshold currents are not used in the circuits discussed in this paper.

One potential problem we should point out with this circuit as shown in the figure is its sensitivity to input voltage noise. Although we are transmitting logical currents, the high impedance decoder input will also respond to voltage noise signals with a high gain. To reduce sensitivity to input voltage noise, we can replace the Widlar-type current mirror circuits with circuits that have a common-gate buffered input, such as the commonly-used cascode- and Wilson-type current mirrors.

In the next section, results of encoder-decoder simulations under on-chip and off-chip interface loading conditions are presented and discussed.

ENCODER-DECODER PERFORMANCE -- SIMULATIONS

The decoder current comparators' current-input--voltage-output DC transfer characteristics are shown in Figure 3. These and all simulation results presented in this paper are obtained using HPSPICE [25] and the model parameters shown in Figure 4 [26]. The model parameter values are for a silicon-gate process that yields 2ns inverter propagation delays for the center inverter in a cascade of three inverters. In Figure 3, we see the current comparators switching at the appropriate threshold current levels with gain that decreases at higher threshold current levels, as we expected. Transmission of binary data through the encoder-decoder combination is illustrated in Figure 5 where we see on top the 2^0 input, below it the 2^0 output, then the 2^1 input, followed by the 2^1 output, and finally the transmitted encoder output current. Transient "glitches" are observed on output waveforms when a change in the input combination is greater than one and thus requires the decoder to temporarily decode the intermediate state. Worst-case propagation delay, when the encoder output current changes three full units of logical current, is simulated to be about 50ns using logical current increments of 10 microamps. The total encoder-decoder power supply current during this transition reaches a maximum of 120 microamps. This delay can be reduced to about 20ns by biasing the encoder and decoder circuits with a small idling current.

By adding a 100pf load capacitance to the line connecting encoder and decoder, we attempted to simulate the environment of chip-to-chip interface through a PC board. Obviously, the small geometry, low current encoder-decoder will operate very slowly under these conditions. Just as in standard CMOS logic design, we need to scale-up the area of off-chip driver devices. Using a four hundred scale factor to develop about 3mA logical currents, worst-case propagation delays were simulated to be about 45ns. With a small idling current biasing the encoder and decoder, this propagation delay can be reduced to about 35ns. One may vary the speed performance of this circuit pair drastically with variations in the logical currents. Since this design uses constant current source circuits, static power dissipation will increase and be proportional to the logical value of the output signal.

SUMMARY AND CONCLUSION

Compatible CMOS binary-voltage--to--quaternary-current encoder and quaternary-current--to--binary-voltage decoder circuits have been described and simulated performance discussed. Encoder current outputs are easily and reliably generated and can be analog summed to perform that arithmetic operation with reduced hardware requirements, but they do suffer from the inability to drive multiple loads. However, in some digital signal processing and interfacing applications single loads are used and current outputs should function well in these applications. Our simulations of this encoder-decoder circuit pair indicate a wide range of propagation delays are available that vary inversely with the magnitude of the logical currents used. The decoder comparators show good discrimination and their thresholds are easily set. Overall, this encoder-decoder pair works well and shows promise. Further development and refinement of these techniques are under investigation.

Given the reduced noise margins of the quaternary logical signals, a well-behaved noise environment would be necessary to minimize transmission errors due to spurious voltage noise. Nevertheless, communication between VLSI chips with quatrnary logic signals could provide important new options for pin-limited chip architectures. Use of the two "extra" states in quaternary logic for testing and diagnosis could also be valuable. Continued investigation into alternatives to binary signal processing, such as MVL, is necessary to be able to take full advantage of the rapidly evolving IC fabrication technologies in the future. Quaternary logic's potential has been widely acknowledged; reduction to practice remains a slow process. These studies attempt to carry the practical consideration of MVL closer to viability.

REFERENCES

1. Mouftah, H.T. and Jordan, I.B., "Integrated Circuits for Ternary Logic," Proc. of the 1974 Intern. Symp. on Multivalued-Logic, pp. 285-302.

2. Vranesic, Z.G., Smith, K.C. and Druzeta, A., "Electronic Implementation of Multi-Valued Logic Networks," Proc. 4th Intern. Symp. on Multiple-Valued Logic, pp. 59-77.

3. Vranesic, Z. and Smith, K.C., "Engineering Aspects of Multiple-Valued Logic Systems," Computer, Vol. 7, No. 9, pp. 34-41, September 1974.

4. Mouftah, H.T. and Jordan, I.B., "Implementation of 3-Valued Logic with C.O.S. M.O.S. Integrated Circuits," Electronics Letters, October 17, 1974, Vol. 10, No. 21, pp. 441-442.

5. Smith, K.C., "Circuits for Multiple Valued Logic: A Tutorial and Appreciation," Proc. 6th Intern. Symp. on Multiple-Valued Logic, May 25-28, 1976, pp. 30-43.

6. Mouftah, H.T. and Jordan, I.B., "Design of Ternary COS/MOS Memory and Sequential Circuits," IEEE Trans. on Computers, pp. 281-288, March 1977.

7. Etiemble, D. and Israel, M., "Implementation of Ternary Circuits with Binary Integrated Circuits," Proc. 7th Intern. Symp. on Multiple-Valued Logic, 1977, pp. 125-132.

8. Liu, T., "Synthesis of Multivalued Feed-Forward MOS Networks," IEEE Trans. on Computers, Vol. C-25, No. 6, pp. 581-588, June 1977.

9. Carmona, J., Huertas, J. and Acha, J., "Realisation of Three-Valued C.M.O.S. Cycling Gates," Electronics Letters, Vol. 14, No. 9, pp. 288-290, April 27, 1978.

10. Koanantakool, H., "Implementation of Ternary Identity Cells using C.M.O.S. Integrated Circuits," Electronics Letters, Vol. 14, No. 15, pp. 462-464, July 20, 1978.

11. Huertas, J., Acha, J. and Carmona, J., "A Note on the Implementation of Three-Valued Unary Operators with CMOS Integrated Circuits," Intern. Journal of Electronics, Vol. 46, No. 2, pp. 205-208, February 1979.

12. Huertas, J.L. and Carmona, J.M., "Low Power Ternary C-MOS Circuits," Proc. 9th Intern. Symp. on MVL, 1979, pp. 170-174.

13. Israel, M. and Entiemble, D., "Some New Results for Ternary Circuits," Proc. 9th Intern. Symp. on Multiple-Valued Logic, 1979, pp. 167-169.

14. Lloris, A., Prieto, A. and Velasco, J., "C.M.O.S. Circuit for Implementation of Unary Operators in Ternary Logic," Electronics Letters, February 28, 1980, Vol. 16, No. 5, pp. 161-162.

15. Etiemble, D. and Israel, M., "On the Realization of Multiple-Valued Flip-Flops," Proc. 10th Intern. Symp. on Multiple-Valued Logic, 1980, pp. 16-23.

16. McClusky, E.J., "Logic Design of MOS Ternary Logic," Proc. 10th Intern. Symp. on MVL, 1980, pp. 1-5.

17. Watanabe, J., Mirua, J., Kurachi, T. and Suetsugut, I., "Seven Value Logic Simulation for MOS LSI Circuits," IEEE Conference on Circuits and Computers, 1980, pp. 941-944.

18. Doa, T.T., "Recent Multi-Valued Circuits," COMPCON Proceedings, February 23-26, 1981, pp. 194-203.

19. Stark, M., "Two Bits per cell ROM," COMPCON Proceedings, pp. 209-212, Spring 1981, San Francisco, CA.

20. Kameyama, M. and Higuchi, T., "Signed-Digit Arithmetic Circuits Based on Multiple-Valued Logic and its Application," Proc. of 1981 Intern. Symp. on Multiple-Valued Logic, pp. 41-53, May 1981.

21. Tront, J.G. and Thakar, A.V., "An Analysis of FET Based Multiple-Valued Logic Circuits," Proc. 12th Intern. Symp. on MVL, 1982, pp. 69-76.

22. Chiang, K.W. and Vranesic, Z., "Fault Detection in Ternary NMOS and CMOS Circuits," Proc. of 1982 Intern. Symp. on Multiple-Valued Logic, pp. 129-138, May 1982.

23. K.W. Current, et al., "VLSI Chip Interfaces with Quaternary Logic," Proc. 1983 Intern. Symp. on Circuits and Systems, May 1983.

24. David A. Freitas and K. Wayne Current, "A CMOS Current Comparator Circuit," IEEE Proceedings, submitted.

25. Hewlett-Packard, "HPSPICE Users Guide," 1982.

26. Les Gehman, et al., "Final Report on 'Modeling for Cellular IC Design'," UCDICL report, October 1982.
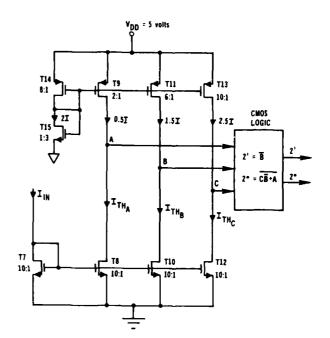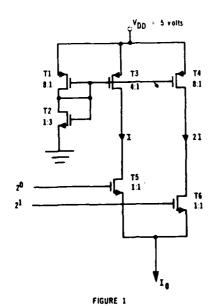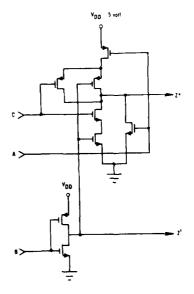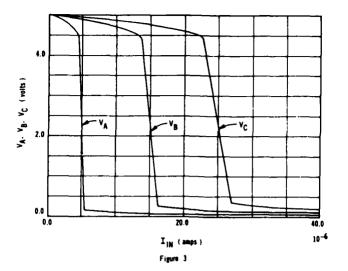
Figure 2a

FIGURE 1
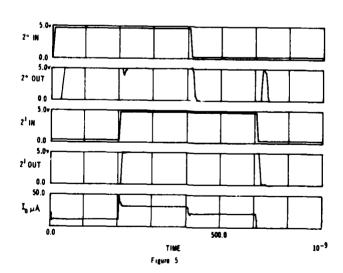
Figure 2b

Figure 3

FIGURE 4

SPICE MODEL PARAMETERS

Figure 5

195

Figure 3

FIGURE 4

SPICE MODEL PARAMETERS

Figure 5

195

REALIZATION AND ANALYSIS OF A MASK-PROGRAMMABLE I²L
MULTIVALUED LOGIC CIRCUIT

Kazutaka Taniguchi[*], Takahiro Inoue[**] and Fumio Ueno[**]

[*]Department of Information and Electronic Engineering,
Yatsushiro Technical College, Kumamoto 866, Japan

[**]Department of Information Engineering, Faculty of Engineering,
Kumamoto University, Kumamoto 860, Japan

Abstract

This paper presents a new mask-programmable I²L multivalued logic circuit. The proposed logic circuit is a variable operational circuit using the ROM structure. The sensitivity analysis is given for this logic circuit assuming equal current gain error for all multicollector transistors and equal current error for all threshold current sources. The limitations on the number of the truth values and on that of the input variables are discussed using the results of the sensitivity analysis. The operation of the proposed circuit is confirmed by the breadboard testing.

1. Introduction

This paper presents a new mask-programmable I²L multivalued logic circuit. Multivalued logic circuits have been constructed by the voltage-mode bipolar or CMOS circuits in the past[1]-[3]. But these circuits required large number of elements and were not suitable for high integration. Recently, I²L technique has been recognized as a promising method for implementing multivalued logic circuits[4]-[5]. For implementing multifunctions in the form of the I²L circuits, the present authors have been proposed a MIN/MAX circuit, a Literal/Successor circuit, a D latch circuit and etc[6]-[8]. In this paper an I²L multivalued universal operational circuit using the ROM structure is proposed. The proposed multivalued operational circuit can provide both the multivalued arithmetic and logic operations by mask-programming the ROM's. Further it has two output terminals, and at each output terminal a different output can be obtained simultaneously. The features of this circuit can be summarized as follows:
(1) high flexibility.
(2) low cost as a result of standardization.
(3) suitability for high integration.
By the sensitivity analysis of the proposed circuit the limitations on the number of the truth values and on that of the input variables are discussed.

2. I²L Multivalued Operational Circuit

Fig.1 shows the general scheme of an I²L multivalued operational circuit using the ROM structure. Logic circuit A is shown in Fig.2. Logic circuits C for the MAX/MIN and the multiplication operation are shown
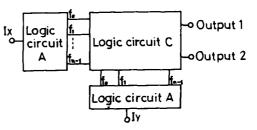
Fig.1 Circuit configuration.
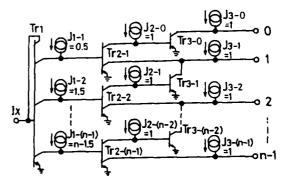
Fig.2 Logic circuit A.
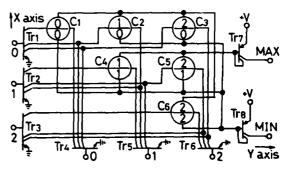
Fig.3 Logic circuit C (MAX/MIN circuit).

196
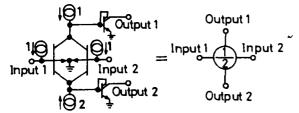
Fig.4 Unit cell (two-input case).
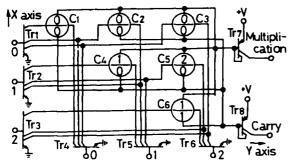
Fig.5 Logic circuit C (Multiplication circuit).

in Figs.3 and 5, respectively. In the logic circuit A of Fig.2, the output for each input Ix is given as follows:

$(f_0, f_1, \cdots, f_{n-1}) = (1, 0, \cdots, 0)$ (for Ix=0)

$(f_0, f_1, \cdots, f_{n-1}) = (0, 1, \cdots, 0)$ (for Ix=1)

$\cdots$

$(f_0, f_1, \cdots, f_{n-1}) = (0, 0, \cdots, 1)$ (for Ix=n-1).

$$(1)$$

Thus the logic circuit A performs as the logic circuit with one input Ix and n outputs f_i's (i = 0 ~ n - 1).
From Eq.(1), f_i can be given as

$$f_i = \begin{cases} 1 \cdots \max(0, i-0.5) \leq Ix < i + 0.5 \\ 0 \cdots \text{elsewhere} \end{cases} \quad (2)$$

where max(a,b) is defined by

$\max(a,b) = a \quad$ if $a \geq b$

$\max(a,b) = b \quad$ if $a < b$.

Evidently from Eq.(1), a following property can be obtained;

$$\sum_{i=0}^{n-1} f_i = 1. \quad (3)$$

As shown in Figs.3 and 5, the logic circuit C of Fig.1 is a current switch array composed of multicollector transistors. Fig.4 shows the unit cell for the crosspoint which is denoted by the circle in Figs.3 and 5. The numerals in each circle denote the magnitudes of the current sources. In Figs.3 and 5, let us name Tr1 ~ Tr3 as X-axis multicollector transistors and Tr4 ~ Tr6 as Y-axis

multicollector transistors. From Figs.3, 4, and 5, it is clear that the value written in the unit cell can be read out only when the X-axis and Y-axis multicollector transistors corresponding to the crosspoint of interest are both in the on-state. The outputs of unit cells in the other crosspoints are zero according to the property of the logic circuit A given by Eq.(3). Then the output of the logic circuit C is obtained by taking "OR" of the outputs of all unit cells. Now if the arithmetic operation with two variables is symmetrical with respect to two variables in logic circuit C, the
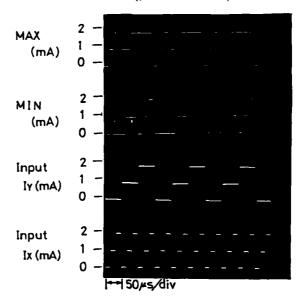
Fig.6 Input/output waveforms of MAX/MIN circuit.
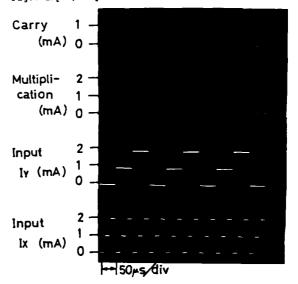
Fig.7 Input/output waveforms of multiplication circuit.

197

	MAX			MIN			Multiplication			Carry		
	Input Ix			Input Ix			Input Ix			Input Ix		
Input Iy	0	1	2	0	1	2	0	1	2	0	1	2
0	0	1	2	0	0	0	0	0	0	0	0	0
1	1	1	2	0	1	1	0	1	2	0	0	0
2	2	2	2	0	1	2	0	2	1	0	0	1

Table 1 Truth value table

required number of the unit cells in the
n × n current switch array can be reduced
to $(n^2 - n)/2 + n$.

In comparison with the two-valued ROM's,
the circuit in Fig.1 can reduce the number
of the input and output terminals as well
as the number of the unit cells. Further
the function of the circuit can be mask-
programmed and two different outputs can
be obtained simultaneously. Since the
logic circuit A in Fig.2 is not so simple,
further simplification is still in study.

Figs.6 and 7 show the input-output
waveforms of the MAX/MIN circuit and of
the multiplication circuit. In the
experiment the three-valued MAX/MIN and
multiplication circuits were tested. The
current values 0, 1, 2 mA were chosen to
the respective truth values 0, 1, 2.
A truth table is given in Table 1.

3. Sensitivity Analysis

3.1 Preliminaries

Let us consider the general I^2L arith-
metic circuit composed of the current
mirrors and the threshold current sources.
Let Ix, Iy, $\cdots$ denote the input currents
or the threshold currents. The symbol δ_β
is the gain error of the current mirror
defined by

$$\delta_\beta \triangleq \frac{I_{in} - I_{out}}{I_{in}} \qquad (4)$$

where I_{in}, I_{out} are the input and output
currents of the current mirror, respec-
tively. The symbol δ_J is the normalyzed
threshold error of the constant current
source defined by

$$\delta_J \triangleq \frac{J - J'}{J} \qquad (5)$$

where J and J' are the design value and
the effective value of the constant
current source, respectively.
Then the output or the effective threshold
current of the arithmetic circuit can be
expressed as the function of Ix, Iy, $\cdots$,
δ_β, and δ_J, which will be denoted by $f(Ix,$
$Iy, \cdots,; \delta_\beta, \delta_J)$. Further in the follow-
ing equations, let us assume $\delta_\beta \gtreqless 0$ and δ_J
can take positive or negative value.
For convenience, the abbreviated forms

$$f(I,\delta) \triangleq f(Ix, Iy, \cdots,; \delta_\beta, \delta_J) \qquad (6)$$

$$f(I,0) \triangleq f(Ix, Iy, \cdots,; \delta_\beta, \delta_J)\big|_{\delta_\beta, \delta_J \to 0} \qquad (7)$$

will be used in the following.
In the arithmetic circuits, let us
further assume the following two condi-
tions:
1. Gain errors of all multicollector
   transistors are equal to δ_β.
2. Normalized threshold errors of all
current sources are equal to δ_J.
The input currents can be assumed to have
no errors without loss of generality.
If the value of $f(I,0)$ corresponding to
the truth value 1 is denoted by f_0, the
relative error M of $f(I,\delta)$ with respect
to $f(I,0)$ is defined by

$$M \triangleq \frac{f(I,\delta) - f(I,0)}{f_0} \qquad (8)$$

3.2 Analysis on the Arithmetic Circuit

For the current source J_{1-k} in Fig.2,
let us denote the designed threshold
value and the deviated threshold value by
J_k and $J_k(1-\delta_J)$, respectively.
For the input value kI_0 of the logic
circuit A, let us denote the deviated
input value by $kI_0(1-\delta_\beta)$, where I_0 is an
output current corresponding to the truth
value 1.
The logic circuit A operates correctly if
the following condition is satisfied:

$$J_{k+1}(1-\delta_J) > kI_0(1-\delta_\beta) \geqq J_k(1-\delta_J) \quad k=1 \sim n-1 \qquad (9)$$

where $J_k \cong (k - 0.5)I_0$.
The relative error of the effective
threshold values (the right or the left
side of Eq.(9)) must be less than a
constant d_1 given by the specification.
Then the sufficient condition for this is

$$(\delta_\beta + |\delta_J|) \hat{J}_n < d_1 \qquad (10)$$

where $J_n > J_{k+1} > J_k$, $\delta_\beta \geqq 0$, $|\delta_\beta - \delta_J| \leqq$
$\delta_\beta + |\delta_J|$, and $\hat{J}_n \triangleq J_n/I_0 = n - 0.5$ were
used.
Since the logic circuit C is a current
switch array composed of multicollector
transistors, the error in the output
currents is directly due to the errors in
the values of the current sources or the
current mirrors. Hence, the relative
error M_c at the output terminal is given
as follows:

$$M_c = \hat{J} - \hat{J}(1-\delta_J)(1-\delta_\beta)^2$$

$$\fallingdotseq \hat{J}(2\delta_\beta + \delta_J) \qquad (11)$$

198

where δ_β, $|\delta_J| \ll 1$, $\hat{J} \triangleq J/I_0$ and J is the current value of the unit cell in Fig.4. $|M_c|$ is restricted by the condition

$$|M_c| < d_2 \qquad (12)$$

where d_2 is a constant provided by the specification, and must satisfy $d_2 \leq 0.5$. From Eq.(12), Eq.(13) can be obtained

$$\hat{J}(2\delta_\beta + |\delta_J|) < d_2 \qquad (13)$$

using $|2\delta_\beta + \delta_J| \leq 2\delta_\beta + |\delta_J|$.

As mentioned above, it is necessary Eqs.(10) and (13) are to be satisfied simultaneously.

Numerical Example

As a numerical example, let us consider the three-valued($n = 3$) two-input case:

$d_1 = d_2 = 0.5$, $|\delta_J| = 0.05$, $\hat{J}_m = 2.5$ and $\hat{J} = 2$.

Then we obtain

$$\delta_\beta < 0.15 \qquad \text{from Eq.(10).} \qquad (14)$$

$$\delta_\beta < 0.1 \qquad \text{from Eq.(13).} \qquad (15)$$

From Eqs.(14) and (15), we obtain

$$\delta_\beta < 0.1. \qquad (16)$$

Therefore, the current mirror with the common base current gain more than 0.9 should be used.

3.3 Limitations

The limitations on the number of the truth values can be obtained by Eqs.(10) and (13).
As a numerical example , let us consider the case with $\delta_\beta = 0.05$, $d_1 = d_2 = 0.5$ and $|\delta_J| = 0.05$.

From Eqs.(10) and (13), we obtain

$$n < 5.5 \qquad (17)$$

and $n < 4.3$, $\qquad (18)$

respectively.
From Eqs.(17) and (18), the maximum number of the truth values is four. Further this circuit configuration can be extended to the multivariable case. In this case the number of the input variables will be limitted by the area of the chip, the complexity of the layout pattern and etc.

Fig.8 shows the scheme of the three-valued four-input operational circuit. The symbol S in Fig.8 denotes the current switch in Fig.9. The symbol C' in Fig.8 denotes the circuit obtained from the logic circuit C by removing the transistors Tr7 and Tr8.

4. Conclusion
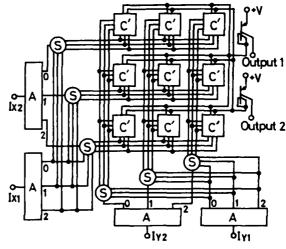
Fig.8 Three-valued four-input operational circuit.
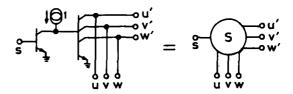
Fig.9 Current switch.

In this paper an I^2L multivalued universal operational circuit using the ROM structure is proposed. The proposed multivalued operational circuit can provide both the multivalued arithmetic and logic operations by mask-programming the ROM's. At two output terminals, different outputs can be obtained simultaneously.
By the sensitivity analysis, the relationship among the tolerances of the multicollector transistors, the number of the truth values and that of the input variables are made clear.
The experiment was carried out using the current mirrors constructed by discrete elements. The performance of the circuit was confirmed up to about 500 kHz. The signal level was set to mA orders.
The problems such as operating speed and power consumption are not treated in this paper. More work is needed on these problems.

References

[1] K.W.Current and D.A.Mow, "Implementing parallel counters with four-valued threshold logic", IEEE Trans. Comput., vol. C-28, pp.200-204, March 1979.

[2] N.Muranaka and S.Imanishi, "Constructions of ternary flip-flop circuits using CMOS-ICs", Trans. IECE, J64-D, pp.445-446, May 1981.

[3] C.Zukeran, G.Hachimine and
M.Shinzato, "Construction of
quarternary logic circuits", Paper
of Technical Group IECE Japan,
CAS81-98, pp.25-30, Feb. 1982.

[4] T.T.Dao, "Threshold I^2L and its
applications to binary symmetric
functions and multivalued logic",
IEEE J.Solid-State Circuits,
vol. SC-12, pp.463-472, Oct. 1977.

[5] J.H.Pugsley and C.B.Sillo,Jr., "Some
I^2L circuits for multiple-valued
logic", Proc. 8th Int. Symp. on
Multiple-Valued Logic, pp.23-31,
1978.

[6] F.Ueno, T.Inoue, K.Taniguchi and
Y.Shirai, "A new multivalued
multifunctional MIN/$\overline{MAX}$ circuit
using I^2L", Trans. IECE, J62-C,
pp.589-591, Aug. 1979.

[7] F.Ueno, T.Inoue and K.Taniguchi, "A
new multivalued multifunctional
Literal/Successor circuit using I^2L",
Trans. IECE, J63-C, pp.199-200,
March 1980.

[8] F.Ueno, T.Inoue and K.Taniguchi,
"New multivalued D latch/D flip-flop
circuits using I^2L", Trans. IECE,
J63-C, pp.312-314, May 1980.

[9] M.Stark, "Two bits per cell ROM",
Proc. of COMPCON Spring, pp.209-216,
Feb. 1981.

[10] C.B.Sillo, "Application of multiple-
valued logic to microprogrammed
processors", Proc. of the 1981 ISMVL,
pp.79-80, May 1981.

[11] K.C.Smith, "The prospects for multi-
valued logic: A technology and
applications view", IEEE Trans.
Comput., vol. C-30, pp.619-634, Sept.
1981.

Logic-type Schmitt circuit using multi-valued gates

by Fumio WAKUI and Masaichi TANAKA

Depertment of Electronic Engineering, Faculty of Science &
Technology, Nihon University, Funabashi City, Chiba, Japan

Abstruct

Logic-type Schmitt circuits (LTSCs) proposed in
this paper by author's proposal are a new detector
for a multi-valued multi-threshold logic circuit,
and it realizes the high resolution with a little
hysteresis or the high noise margin. The detector
consists of the conbinations of the multi-valued
gates (MVGs) and a positive reaction device (PRD),
and each circuit can be realized by the convention-
al circuit elements.

This paper shows their practical circuits, and
describes the regions and the conditions for their
operation.

1. Introduction

It is well known that Schmitt circuits detect an
analog input levels and obtain a digital output
with the waveform-sharping under the high resolu-
tion or the high noise margin conditions. The positve
feed back circuits corresponding to their circuits
are used as the logic circuits with the high noise
margine in digital systems. However, it is difficult
to realize the large logic circuits because of the
circuit and logic analyses for it. But, the LTSCs
will realize a larger circuit than them.

As the input-output characteristics of the LTSCs
depend directly on the graphical presentation which
is shown by a transformation of a logic function[1]
,[2]. they have transfer characteristics with pros-
pects for less circuits and logic analyses etc. by
comparison with the conventional ones.

The construction methods by the practical cir-
cuits of the LTSCs use a combination of the MVGs real-
lized by the conventional elements with a reference
terminal and a PRD [3] with a special response in
regard to the current inputs. As the LTSCs in this
way are constructed by the simple couplings of the
MVGs and a PRD, the sufficient temperature stabil-
ity and the permissible error for the elements can
be provided with a few considerations. Further, the
minimum gate-current and the operational region for
the LTSCs are described and investigated by the
comparison with the experimental circuits in this
paper. As the future problems, when the LTSCs shall
be attempted for the practical applications [4]-[6],
then LTSCs will gain their position so that they are
required for the binary logic or multi- valued logic
circuits.

2. Physical meanings and Definitions

2.1 Physical meanings

A r-valued v-threshold logic function is known
with the interesting contents by[1],[2], in which it
realizes a single output for the many inputs. A log-
ic input vector X $(x_1,...,x_n)$ of it generates an
excitation "e" by a transformation of a logic func-
tion with a proper weight vector $W(w_1,...,w_n)$. The
"v" is chosen according to Eqn.(1) so that the "e"
shall be taken the different values for different
truth values. And the "e" produce a logic output "y"

$$e = W \cdot X \qquad (1)$$

classified by a multi-threshold detector with a
threshold vector $T(t_1...,t_v)$. Here, the "y" against
the "X" can be presented by a value vector $Y(y_1,...
,y_k,...,y_{v+1})$ of the logic values according to the
transformation, so that it is defined by the follo-
wing equation.

$$y \ni y_k \qquad (2)$$
$$k = 1,2,...,v+1$$
$$y = (0,1,2,...,r-1)$$

2.2 Definitions and operation for the LTSCs

This paper describes about the construction

(a)

(b)
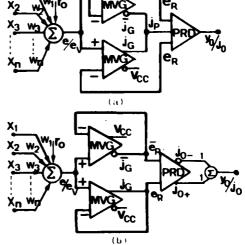
Fig. 1 Practical models of the LTSCs

201
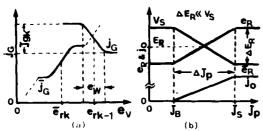
Fig. 2 Input-output characteristics of
the MVGs and the PRD

methods of the LTSCs based on the preceding physical
meanings. And it is almost expended by the descrip-
tions in which a new detector is realized.

For constructing the practical models by our pro-
posal in Fig.1, the following symbols are used.

(1) A voltage signal e_v and current signal j_o
are defined as the circuit symbols
corresponding to the "e" and "y".

(2) The currents $(J_1,...,J_k,...,J_{v+1})$ correspond-
ing to the vector "Y" are defined.

(3) A excitation voltage ϕ_u with a minimum unit
is defined as a minimum distance between
the each thresholds.

(4) The gate-currents $(J_{gk},\overline{J}_{gk})$ which are pres-
ented by the polarities and a magnitude of
the $(J_{k+1}-J_k)$ are defined.

(5) A $(\phi_w)_{gk}$ with a magnitude of the $(J_{gk},\overline{J}_{gk})$
is defined as a magnitude of the
differential voltage in regard to the e_v
with the hysteresis.

(6) A DC voltage E_{rk} which correspods to a value
of the threshold "t_k", and the reference
signals $(e_{rk},\overline{e}_{rk})$ in which the E_{rk} and the
two signals $(\triangle e_R,\triangle\overline{e}_R)$ corresponded to a
variation of the e_v are included, are
defined.

(7) A DC voltage E_R from which the E_{rk} is
supplied, and the response signals $(e_R,\overline{e}_R)$
from which the levels of the $(e_{rk},\overline{e}_{rk})$ are
shifted, are defined.

(8) The $(\Delta E_R,\Delta\overline{E}_R)$ are defined as the maximum
voltages of the $(\triangle e_R,\triangle\overline{e}_R)$, and a signal ΔJ_p
corresponded to the $(\Delta E_R,\Delta\overline{E}_R)$ is determined.

In Fig.1(a) the e_v of the detector with an analog
or a multi-valued input becomes an input-current
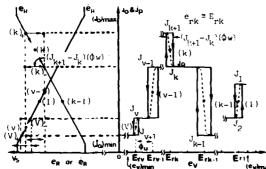
signal j_p of a PRD by a wired summation of the cur-
rents $(j_G,\overline{j}_G)$ judged by the multi-reference voltages
of the MVGs. As a PRD has an alteration quantity
Δj_p of the j_p. The response signals $(e_R,\overline{e}_R)$ generate
the varieties $(\triangle e_R,\triangle\overline{e}_R)$ corresponding to them. And
the $(e_R,\overline{e}_R)$ are distributed the multi-reference
signals $(e_{rk},\overline{e}_{rk})$ by the level shifts in the MVGs,
so that they supply as the positive feed backs for
the reference voltages.

In Fig.1(b), A special connection between the j_G
and the e_v ensures to operate at smaller gate-cur-
rents than another. They are designed so that, dur-
ing the operations of the LTSCs, the $(\Delta E_R,\Delta\overline{E}_R)$ of a
PRD can be ignored against a DC voltage E_R in Fig.2
(b). But their conditions for the LTSCs with a bi-
nary output can be excluded. Accordingly, in Fig.2
(a), when the E_R is used as the reference voltages
of the MVGs, the $\overline{j}_G$ can be shown against the j_G. And
at the time of the perfect cut-over of the $(J_{gk},\overline{J}_{gk}$
) of the MVGs, the e_w is required for the e_v.

In Fig.2(b), the linearity and stability of a
output signal j_o and the $(e_R,\overline{e}_R)$ corresponded to a
variation of the input signal $(j_p$ or $j_{p+},j_{p-})$ of a
PRD are required.

Fig.3 shows the j_o of the LTSCs obtained by the
mutual operation between the MVGs and a PRD. Now,
when a input terminal of the detector by the MVGs
has the e_v, the j_o can be shown with a general form
as a dotted line at the right of the same figure.
And at the left of the same figure, a PRD against
the e_v generates an output and the $(e_R,\overline{e}_R)$
corresponded to the points shown in the j_o.

Next, as a loop current gain at the all regions
of the transitions of the LTSCs is 1 or more, they
have a hysteresis characteristic in proportion to
the J_{gk} at the points (k and v) shown as a sample.
And, near the point (k), when the j_o of the LTSCs
changes from the J_{k+1} to the J_k by increase of the
e_v, then in order that the e_R is kept to the point
$(k)_+$, the holding are executed so that the e_R
against the E_{rk} is too large by the $\phi_w(J_{k+1}-J_k)/2$.
Here the ϕ_w is a quantity corresponded to hysteresis
with a unit current. While the restraints with the
inverted holding are applied at the time of decrease
, so that a hysteresis corresponded to the $\phi_w(J_{k+1}-$
$J_k)$ in the same figure is caused. In the same way,
inverted hysteresis is caused near the (v).

Fig. 3 Mutual operation between the MVGs
and the PRD
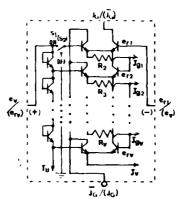
Fig. 4 Practical circuit for the MVGs
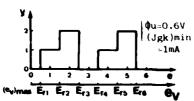
Fig. 5 Graphcal presentation as example of the MVGs
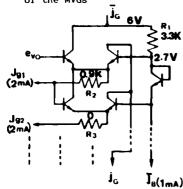
Fig. 6 Special operation of the MVGs

3. Practical circuit and operation conditions

The LTSCs shown in Fig.1 consist of an weight circuit with no feed back and a multi-threshold circuit which has the positive feed back loops from a PRD to the MVGs. And, the LTSCs execute some logic functions by the set of the resourceful weights against the inputs $(x_1,...,x_n)$ and with the several thresholds, in which the multi-threshold circuit consist of the MVGs obtained by the vertical-cascade connections and a PRD with a common base connection at the terminal of the j_p, so that it will have a high speed characteristic.

To ensure the stable operation for a realization by the circuits, a circuit design under a consideration of temperature characteristics is important. The LTSCs are constructed according to the following conditions.

(1) The logic inputs $(x_1,..,x_n)$ shall consist of the currents with the sufficient stability, the all currents shall be supplied from a multi-current source.

(2) All internal reference voltages of the MVGs shall be decided by the resisters $(R_2,...,R_k, ...,R_v)$ and the current sources $(J_{g1},..,J_{gk}, ...,J_{gv})$.

(3) The signals $(e_R,\bar{e}_R)$ shall have DC voltage component E_R with the sufficient stability.

Fig.4 shows a practical circuit for the MVGs. When the e_R is supplied to the (-) terminal, one of the MVGs decides the many internal reference voltages $(E_{r1},..,E_{rk},...,E_{rv})$ by the voltage drop from the resistances $(R_2,...,R_k,...,R_v)$ and the currents $(J_{g1}, ...,J_{gk},....,J_{gv})$. Here, the R_k is decided by the J_{gk} and the differencial voltage $(E_{rk-1}-E_{rk})$.

For example, now we are planning to design for the realization of a graphical presentation shown in Fig.5. And as the circuit conditions, when a value of the ϕ_u and a minimum gate-current $(J_{gk})_{min}$ are given by 0.6V and 1mA, one of the MVGs has the resistances $(R_2=R_4=0.6k, R_3=1.2 k)$ when the switch S_1 is on. For another the resistances
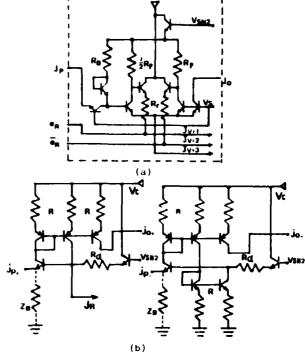
Fig. 7 Practical circuits for a PRD
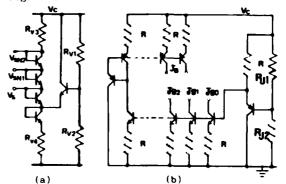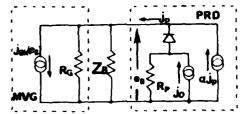
Fig. 8 Voltage and current sources

are $R_2=0.6k, R_3=0.9k$ when the S_2 is off, because, the current J_{gk} in this MVGs can use the quantity (2mA) with the $2(J_{gk})_{min}$ in the same figure. Further, as a special operation, when the E_R is supplied to (+) terminal, the $\overline{MVGs}$ operate by the levelshifts of the e_v, and the E_R is adjusted to the E_{rv}. At this time, a relation of the complementary outputs $(j_G \& \bar{j}_G)$ inverts, and an unshown resistance R_1 must be used between the $\overline{MVGs}$ and the PRD so that the differencial voltage $((e_v)_{max} -E_{r1})$ can be gained by the R_1 and a costant current J_B. For example, the R_1 for the $\overline{MVGs}$ with the $((e_v)_{max} \simeq V_s=6V$, and $J_B =(J_{gk})_{min}$ against the j_G) becomes $(0.6(6-1)+0.3)/1=3.3k$ when the S_2 is on in Fig.6. Fig.7 shows two types of the practical PRD. The type in Fig.7(a) puts out the response signals $(e_R,\bar{e}_R)$ corresponding to the E_R and the $\triangle e_R$ and the level of the $(e_R,\bar{e}_R)$ is shifted by the $(J_{v+1}.J_{v+2})$ the j_p, and resistances. Here, in the same figure, a small resistance R_r must be chosen according to the $R_r.J_{v+1}, R_r.J_{v+2} \simeq \phi_t$ so that the PRD can be satis-

Fig. 9 Presentation of the LTSCs for the loop gain

fied the condition of the unsaturation for the MVGs. And, when the ΔE_R and the ΔJp were given as the operation range of a PRD, the $R_B \cdot R_p \cdot J_{v+1}/(2e_w)$ must be designed to become the ratio ($\Delta E_R/\Delta Jp$).

For example, at the ($\Delta Jp=5mA, \Delta E_R=0.6V, e_w=0.15V$, and $J_{v+1}=J_{v+2}=1mA$). the $R_B \cdot R_p$ becomes 0.036k, so that the R_B at the $R_p=0.5k$ must use 0.072k. Fig.7(b) shows a circuit with a loose coupling condition for the MVGs. For a PRD in this case, it is necessary that the R_α in same figure shall be chosen by the ($\Delta E_R/\Delta Jp$), and no other conditions are necessary in order to operate in the wide range of the j_p.

Fig.8 shows a voltage source and a current source used for the LTSCs. The voltage source shown in Fig. 8(a) can supply V_{sn1}, V_{sn2} with a negative temperature coefficient and the V_S with the sufficient stability in regard to the temperature. With $\Delta K_B=1+\Delta\beta t/\beta t$ and $\Delta K_\phi t=1-\Delta\phi t/\phi t$ as the coefficients of temperature variations for the transister current amplification factor βt and the forward voltage ϕt, a constant $m=\Delta K_\phi t/\Delta K_{\beta t}$ can be established as an approximation over a wide range. Using these, the stability condition for the V_S can be expressed as follows;

$$2m\frac{V_{\beta to}}{V_c}\phi o(\frac{R_{v3}}{R_{v4}})^2+V_{sc}(\frac{R_{v3}}{R_{v4}})+V_{sc}-2m\frac{V_{\beta to}}{V_c}\phi o-V_c=0$$

At this time, the V_S is obtained by Eqn.(3).

$$V_S=V_{sc}+2m(\frac{R_{v3}}{R_{v4}}-1)\frac{V_{\beta to}}{V_c}\phi o \qquad (3)$$

with $\beta o, \phi o$: βt and ϕo at room temperature.

$$V_{\beta to}=\frac{R_{v1}//R_{v2}}{\beta o R1}V_c, \qquad V_{sc}=\frac{R_{v1}//R_{v2}}{R_{v1}}V_c$$

3.1. Loop gain and basic design of the LTSCs

The LTSCs are designed so that a loop gain between the MVGs and a PRD becomes 1 or more. The following definitions are made from the MVGs and a PRD characteristics of Fig.2. Next, a positive feed back

Definitins; $\alpha_R=de_R/dj_p$, $(\alpha_c)_k=de_v/dj_{gk}$
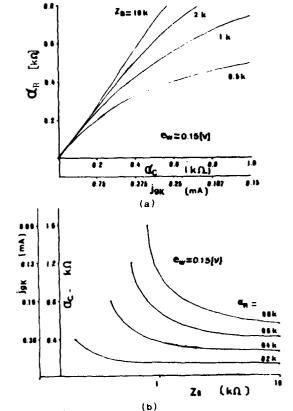
model for the LTSCs is shown in Fig.8. According to Fig.9, the circuit loop gain G can be shown by Eqn.(4). The G for a PRD of Fig.7(a) is obtained in the same way, and it corresponds to the case of $Z_B \to \infty$

$$G = \frac{j_{gk}(e_R)+e_R((j_p)_k)/Z_B}{(j_p)_k}$$

$$= \alpha_R/(\alpha_c)_k+\alpha_R/Z_B \qquad (4)$$

with $e_R((j_p)_k)=\alpha_R(j_p)_k$

$j_{gk}(e_R)=e_R((j_p)_k)/(\alpha_c)_k=\alpha_R(j_p)_k/(\alpha_c)_k$

in Eqn.(4). The LTSCs for the small current of the MVGs can be designed with the large α_R or the small Z_B from the definition and Eqn.(4), but our objects

(a)

(b)

Fig. 10 Specification of the PRD and the MVGs for the LTSCs

are not to choose a best method.

When the LTSCs in Fig.1(a) are applied to the realization of an r-valued logic function, the magnitude of the ΔJp shows with the $(J_{gk})_{min}$ and the "r" by Eqn.(5). Accordingly, $\alpha_R \cdot \Delta Jp$ is required for the ΔE_R.

For example, now the LTSCs are used in order to realize a quaternary logic function, and when the conditions of the construction and the operation are shown by the ($G=1.5, (J_{gk})_{min}=1mA, e_w=0.15V, Z_B=1k$), then the α_R according to Eqn.(4) becomes 0.196k. Accordingly, a PRD must be designed so that the ΔJp and the E_R have 3.73mA and 0.73V.

$$\Delta Jp = (r-1)(J_{gk})'_{min}$$

$$= (r-1)Z_B(J_{gk})_{min}/(Z_B-\alpha_R) \qquad (5)$$

With $(J_{gk})'_{min}=(1+(\alpha_R/Z_B)+(\alpha_R/Z_B)^2+\dots$

$$\dots+(\alpha_R/Z_B)^{po})(J_{gk})_{min}$$

$$Z_B > \alpha_R$$

For the LTSCs, the magnitude of Eqn.(6) can be expected for the $(\phi_w)_{gk}$ from the definition and Eqn.(4). For designing some logic circuit by the LTSCs shown in Fig.1(b), the ΔJ_p and the ΔE_R with the large values for a PRD can be expected and their magnitudes can be obtained by Eqn.(5) and a value of the logic levels in the detector in which it shall be decided by the number of the threshold and the shape of the step-slope in Fig.3. Next, when a condition

$$(\phi_w)_{gk} = \frac{Z_B \cdot e_w \cdot G}{Z_B-(G-1)\alpha_c} \qquad (6)$$

204
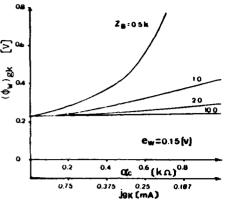
Fig. 11 Magnitudes of the Hysteresis under the conditions
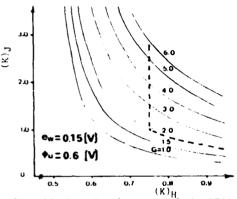
Fig. 12 Operational region of the LTSCs with binary output

Fig. 13 The hysteresis and the decrease of the current as experimental result
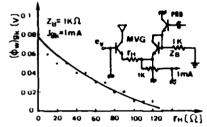
Fig. 14 Decrease of the practical value for the $(\phi_w)_{gk}$ by the r_H

with the $\Delta J_p \leq (J_{gk})_{min}$ is included as the conditions for the design, then they generate only a binary output. Eqn.(6) isn't effective, and the $(\phi_w)_{gk}$ can not be presented for their LTSCs. Accordingly, it is necessary to present their relations so that their conditions are fulfilled only for the application from the binary to LTSCs. And, from the following definition and Eqn.(4), the hysteresis characteristics can be presented by Eqn.(7). Eqn.(7) is effective for the $(K)_J$ with the all regions, but when the $(K)_J$ is 1 or less, the $(\phi_w)_{gk}$ for their LTSCs can be led from a multiplication of the $(K)_J$ and the ΔE_R shown in the definition $(K)_H$.

$$(K)_J = (J_{gk})_{min}/\Delta J_p \quad , \quad (K)_H = (\phi_u + \Delta E_R)/2\phi_u$$

$$(K)_H = \frac{1}{2}\left(\frac{e_w}{\phi_u(K)_J} G + 1\right) \qquad (7)$$

The results obtained from a few analyses for the preceding LTSCs are shown in Fig.10 to Fig.12. Fig. 10 shows the LTSCs construted by a PRD which all fulfill Eqn.(4), and the G is 1.5. Fig.10(a) shows the specifications for a PRD, and the design of the α_R under the loose conditions by Z_B is shown. Fig.10 (b) shows the reduction of the $(J_{gk})min(=e_w/(\alpha_c)_k)$ of the MVGs by Z_p with the α_R as parameter and the minimum value. Fig.11 shows the variation change of the $(\phi_w)_{gk}$ obtained by Eqn.(6). From the precedence, the multi-valued response in the range of $\Delta J_p > ($

$(J_{gk})_{min}$ requires the operation of the LTSCs for a still smaller $(J_{gk})_{min}$, and at this time, a linear increase for the α_R of a PRD or decrease of Z_B is required. In regard to the latter decrease of Z_B, a slight increase of the $(\phi_w)_{gk}$ can be expected.

Fig.12 shows that a circuit construction with a unique characteristic and a wide operation range in regard to the binary response of the LTSCs can be realized. For example, when the LTSCs are designed with the $(\phi_w)_{gk}=0.3V$ against the $\phi_u=0.6V$, the choice of the region shown on the dotted line in same figre can be permitted, because the $(\phi_w)_{gk}$ at the $(K)_J \leq 1$ can be shown by Eqn.(6), and the $(\phi_w)_{gk}$ is constant at the range of the $G \geq 2$ against the $(K)_J \geq 1$.

In regard to either response, the minimum hysteresis is limited by the e_w of the MVGs, and a realization of the circuit with a still smaller $(\phi_w)_{gk}$ can be reached by insertion of a small resistance r_H at the ECL emitter couplings of the MVGs.

3.2 Experimental results

The many experimental circuits based on the preceding analyses for the LTSCs have been investigated, and here, some results of them are shown.

When the LTSCs have been tested so that the G are keeping to 1.5 as a theoretical value, the variations of the $(\phi_w)_{gk}$ against the Z_B and the J_{gk} are shown as the experimental results in Fig.13. By comparison with the theoretical $(\phi_w)_{gk}$ in Fig.11, the $(\phi_w)_{gk}$ at
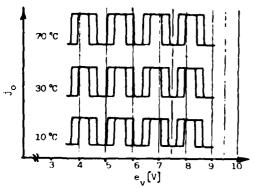
Fig. 15 Independence property of the thre-
shold voltages from temperature
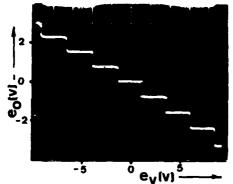
Fig. 16 Waveform of a Schmitt circuit for the
symmetric signal by the LTSCs

the $Z_B = 1k$ have the larger values than the other.
Because, the theoretical $(\phi_w)_{gk}$ on near here generate
the larger variations and the practical $(\phi_w)_{gk}$ are
generated by the parameters with the permissible
errors.

Next, at the $G=1.5$ with $Z_B=1k$ and $J_{gk}=1mA$ in
Eqn.(4). the effect for the $(\phi_w)_{gk}$ against the r_H
have been investigated, and their results are shown
as the decrease of the $(\phi_w)_{gk}$ in Fig.14. Further, it
shows that the r_H can be chosen so that the $(\phi_w)_{gk}$
becomes few values from the same figure.

Next, the temperature independency is for the
multi-threshold voltage of the LTSCs have been
investigated. The voltage source and the current
source in Fig.8 have been tested so that a 2-valued
8-threshold circuit was constructed by the MVGs and
a PRD in Fig.7(a). The temperature characterisyic
for the V_s of the voltage source with the ($R_{V1}=3.3k$,
$R_{V2}=10k$, $R_{V3}=R_{V4}=1.5k$) in Fig.8(a) has been presented
by $1.0mV/°C$ against the voltage supply V_c from 7V to
12V. And the temperature coefficient of the multi-
current source with the ($R=0.75k$, $R_{J1}=10.6k$, $R_{J2}=0.68k$
) in Fig.8(b) has been presented by $390ppm/°C$.

For a 2-valued 8-threshold circuit constructed by
them, A maximum variation against the temperature in
the threshold voltages has been obtained by the
lowest threshold voltage, and the magnitude has been
presented by $1.5mV/°C$ in Fig.15.

At the first, a schmitt circuit has been con-
structed so that it executed the detection and the
waveform-shaping for a input with a symmetrical
signal against the ground. And the input-output

waveform is shown in Fig.16.

4. Conclusion

It has been shown that the LTSCs can be realized
easily by a simple principle in case of the MVGs
constructed by the elements (ECL,SCL) with a
reference terminal. The stable operation of the
threshold voltages and the logic output levels of
the LTSCs over a wide temperature range have been
confirmed.

The LTSCs will advance the increase of the noise
margin or the resolution for the detector of a
multi-valued multi-threshold logic circuit, and the
new circuit realization for an application with the
different region in regard to the conventional
region will be expected. And in this paper as an
investigation for the integrated circuits of the
LTSCs, the LTSCs have been constructed so that the
decrease of the gate-current and the large or small
quantity of the $(\phi_w)_{gk}$ were attempted to the cir-
cuits for the practical applications. And, by the
analyses and the experimental circuit, the reasons
stopping their attempts could not be found.

In regard to the practical investigation for the
packing density, the operation speed, and the noise
margin etc. When the LTSCs for the applications in
the regions with the grate effects will be used,
their magnitudes must be investigated as the prac-
tical values.

REFERENCES

(1) O.Ishizuka:"Synthesis of muti-valued muti-
     threshold network",IECEJTD,J60-D,6,pp459,
     (June.1977)

(2) A.Druzeta,Z.Z.Vranesic and A.S.Sedra:"Applica-
     tion of multi-threshold elements in the reali-
     zation of many-valued logic networks",IEEE TC,
     C-23,11,pp1194,(Nov.1974)

(3) F.Wakui and M.Tanaka:"Constitution of arbi-
     trary multi-valued Schmitt circuit",IECE,J65-C
     ,2,pp130,(Feb.1982)

(4) M.Kameyama and T.Higuchi:"Signed-digit arith-
     metic circuits based on multiple-valued logic
     and its applications",proc.ISMVL,pp41,(1981)

(5) K.W.Current:"A simultaneous analog to quater-
     nary converter",IEEE.Trans on comput and
     system, vol CAS-26, No.11,(Nov.1979)

(6) M.Briman,D.Etiemble,J.L.Oursel, and P.Tatareau
     :"A 4-valued ECL encoder and decoder circuit",
     IEEE J.of solid-state circuit,vol.SC-17,No.3,
     pp547,(June 1982)

Session 5B
Philosophy II

A GENERAL METHOD FOR THE EVALUATION OF DEGREE OF COMPLETENESS

by Alan Rose

Department of Mathematics, University of Nottingham, England

Abstract

Substitution rule type formalisations of propositional calculi without constants are derived from existing axiom scheme type formalisations. The degree of completeness is evaluated in the case where the variable functors take values in the set $\{C, C'\}$ of Łukasiewicz m-valued or $\aleph_0$-valued implication and its converse. A generalised result is obtained for the cases where the degrees of completeness of the corresponding formalisations of conventional m-valued and $\aleph_0$-valued propositional calculi are known. A method for the evaluation of these latter degrees of completeness in a wide class of cases is also given.

We shall consider first m-valued ($m=2, 3, \ldots$) and $\aleph_0$-valued propositional calculi whose only primitive symbols are propositional variables p, q, r, ... and binary variable functors $\delta, \varepsilon, \lambda, \ldots$ having values in the set $\{C, C'\}$, where C denotes the implication functor of Łukasiewicz [3] and C' denotes the corresponding converse implication functor. The syntactical variables P, Q, ...; Δ, Ω, ... will be used to denote formulae and variable functors respectively. These calculi have been formalised by the author [4] and Jones [1] by means of a finite number of axiom schemes and the following rule of modus ponens, where

$$C_\Delta PQ =_{df} \Delta \Delta QP \Delta PC$$

(so that $C_\Delta PC =_{\tau} CPC'$):

R1 If P and $C_\Delta PQ$ are correct formulae then Q is a correct formula.

We may obviously obtain new weakly complete and plausible formalisations $\mathcal{F}_m$ ($m=2, 3, \ldots$), $\mathcal{F}_{\aleph_0}$, using only finitely many axioms, if we replace the syntactical variables P, Q, R, Δ, Ω, Λ of the axiom schemes (but not of R1) by p, q, r, $\delta, \varepsilon, \lambda$ respectively (so that, for example, the axiom scheme A1 of the author's previous paper becomes the axiom $C_\delta p C_\varepsilon q p$, i.e. $\delta\delta\delta p c\delta_\varepsilon pp\delta p\delta\delta pq\delta qp$) and adjoin the two following primitive rules of procedure:

R2 If P is a propositional variable occurring in the formula P and the result of replacing all occurrences of p in P by the formula R is Q then, if P is a correct formula, Q is a correct formula.

R3 If Δ is a variable functor occurring in P and the result of replacing all occurrences of Δ in P by the variable functor Λ is Q then, if P is a correct formula, Q is a correct formula.

R1 may be replaced by the following rule R1A, since R1 follows from R1A and R3:

R1A If P and $C_\delta PQ$ are correct formulae then Q is a correct formula.

(Let P' be the result of replacing all occurrences of δ in P by a variable functor Λ which does not occur in $C_\Delta PQ$ and let P* be the result of replacing all occurrences of Δ in P' by δ. Let Q', Q* be defined similarly. Thus P, $C_\Delta PQ \vdash P'$, $C_\Lambda P'Q'$ and P', $C_\Delta P'Q' \vdash P*$, $C_\delta P*Q*$ by R3. Using R1A and R3 we then infer Q*, Q' and Q.)

The question of the values of the ordinal[15,9] degrees of completeness of the new formalisations then arises and we shall establish a theorem for a class of formalisations which includes $\mathcal{F}_m$ ($m=2, 3, \ldots$) and $\mathcal{F}_{\aleph_0}$. We shall then consider, in a more general way, the relationship between the degree of completeness of a formalisation of a propositional calculus without constants and the degrees of completeness of the related conventional formalisations. Finally we shall give a method for the evaluation of these latter degrees of completeness which applies to a wide class of cases.

Theorem. If formalisations of an m-valued ($m=2, 3, \ldots$) and an $\aleph_0$-valued propositional calculus without constants corresponding to the set $\{C, C'\}$ are weakly complete and plausible and the only primitive rules of procedure are the rules R1A, R2, R3 then the formalisations have degrees of completeness $3m-2$ and ω respectively.

We note first that, in the m-valued case, using the notation of a previous paper of the author [5] concerning the corresponding problem where there are no variable functors and

C is the only primitive functor,

$$\xi(CC(Cp)^{k-1}Cpp)=k \quad (k=1, \ldots, m)$$

and the formula $C'C'(C'p)^{k-1}Cpp$ takes the same truth-value as CpCpCCp or as CpCpCCCpp according as k is even or odd. Thus the formula $\delta\delta(\delta p)^{k-1}Cpp$ takes the same truth-value as a formula P_k for which $\xi(P_k)=k$ when δ takes the value C and it always takes the truth-value 1 when δ takes the value C'. It follows at once that, for the formula $\delta p\delta p\delta^{k-1}Cp \ldots p$, the roles of C and C' are interchanged. Let us abbreviate the latter formula and its counterpart by Y_k, X_k respectively. We note also that if Z_k denotes the formula

$$\delta\delta\delta pp\delta(\delta p)^{k-1}C\varepsilon\delta pp$$

then Z_k takes the truth-value 1 unless δ, ε take the values C, C' respectively, in which case it takes the same truth-value as a formula P_k $(=CCCpp(Cp)^{k-1}pC'pCpp=CC(Cp)^{k-1}Cpp)$ for which $\xi(P_k)=k$. Let P', P'' be the formulae obtained from an arbitrary formula P by replacing all occurrences of variable functors by C, C' respectively and let P''', ..., $P^{(2^\alpha)}$ be the remaining formulae obtained by replacing all occurrences of variable functors by C and C', each occurrence of the same variable functor being replaced by the same constant functor, where α is the number of distinct variable functors occurring in P. Let

$$\Omega(P)=\min(\xi(P'), \ldots, \xi(P^{(2^\alpha)})),$$

$$\equiv(P)=(\xi(P'), \xi(P''), \Omega(P)).$$

We partially order the ordered triples $\equiv(P)$ by requiring that

$$\equiv(P) \preceq \equiv(Q)$$

if and only if

$$\xi(P') \le \xi(Q'), \ \xi(P'') \le \xi(Q'') \text{ and } \Omega(P) \le \Omega(Q).$$

We then define the relation "$\prec$" in the obvious way.

Lemma. If $\equiv(P_i)=(a_i, b_i, c_i)$ ($i=1, \ldots, n$), $A=\min(a_1, \ldots, a_n)$, B, C being defined similarly, a necessary and sufficient condition that $P_1, \ldots, P_n \vdash Q$ is $(A, B, C) \preceq \equiv(Q)$.

Proof of necessity. Let l denote the total number of applications of R1A-3 in the derivation of Q. We shall prove the result by strong induction on l. If l=0 then Q is an axiom, in which case $\equiv(Q)=(m, m, m)$, or Q is one of $P_1, \ldots, P_n$ in which case, for some $i \in \{1, \ldots, n\}, \equiv(Q)=(a_i, b_i, c_i)$ and $a_i \ge A$,

$b_i \ge B$, $c_i \ge C$. In both cases the result follows at once.

We now assume the result for 0, ..., l-1 and deduce it for l. If the last step in the derivation of Q is an application of R1A to two formulae R, $C_\delta R$ and, under a particular assignment of values to the variable functors of Q, R, the formulae R, $C_\delta R$ take the same truth-values as formulae R', CR'Q' respectively, then

$$\xi(R'), \ \xi(CR''') \ge C$$

so that $\xi(Q') \ge C$. Thus the last member of the ordered triple $\equiv(Q)$ is at least C. If all variable functors are assigned the value C (C') then

$$\xi(R'), \ \xi(CR''') \ge A(B),$$

so that $\xi(Q') \ge A(B)$ and the first (second) member of the ordered triple $\equiv(Q)$ is at least A(B). Hence $(A, B, C) \preceq \equiv(Q)$.

If the last step is an application of R2 to a formula R then, using the notation of the last paragraph, $\xi(R') \ge C$ and, since Q' is obtainable from R' by substituting for a propositional variable, $\xi(Q') \ge C$. In the two special cases it follows similarly that $\xi(Q') \ge A(B)$ and the result follows at once.

If the last step in the derivation of Q is an application of R3 to a formula R and $2^{\alpha(Q)}=\beta$, $2^{\alpha(R)}=\gamma$ then

$$\{R', \ldots, R^{(\gamma)}\} \supseteq \{Q', \ldots, Q^{(\beta)}\},$$

so Q'=R', Q''=R'', $\Omega(Q) \ge \Omega(R)$ and the result follows at once.

Proof of sufficiency. We may suppose that, without assuming λ, μ, τ distinct the first, second and third members of the ordered triples $\equiv(P_i)$ are least for P_λ, P_μ, P_τ respectively. Let K be chosen from the set $\{1, \ldots, 2^{\alpha(P_\tau)}\}$ in such a way that $\xi(P_\tau^{(K)})$ is least. It will be convenient to define (cf. [1]) the summation operators Γ, $_\delta\Gamma$ with respect to C, C_δ respectively. We may, as in [5], infer from P_λ' and the classical rule of substitution for propositional variables, formulae $P_{l+1}, \ldots, P_d$ such that the formula

$$\Gamma_{i=l+1}^d P_i Q'$$

always takes the truth-value 1. (The formulae P_{l+1}, ..., P_d may not be distinct, but it will not always be necessary to repeat each formula m-1

times, corresponding to the definition of a
standard condition implication functor by $(m-1)$-
fold iteration of the functor C, cf. [8].) Thus,
if Δ is a variable functor of Q and $P*_i$ is
obtained from P_i by replacing all occurrences of C
by Δ ($i = n+1, \ldots, d$), the formula

$$ {}_\delta \Gamma^d_{i=n+1} P*_i Q $$

will take the truth-value 1 whenever all variable
functors of Q take the value C. Similarly we may
construct a formula

$$ {}_\delta \Gamma^e_{i=d+1} P*_i Q $$

corresponding to the case where they all take the
value C' (using P_μ' in place of P_λ').

If $\gamma \in \{3, 4, \ldots, 2^{\alpha(Q)}\}$ we may similarly,
since $(A, B, C) \leqslant \equiv(Q)$, infer from $P_\gamma^{(K)}$ and the
classical rule of substitution for propositional
variables, formulae $P_{e+1,\gamma}, \ldots, P_{f\gamma}$ such that the
formula $\Gamma^f_{i=e+1} P_{i\gamma} Q^{(\gamma)}$ always takes the truth-value
1. Since $\gamma \neq 1, 2$ we may choose variable functors
$\Delta_\gamma, \Lambda_\gamma$ which are replaced by C, C' respectively
in the construction of $Q^{(\gamma)}$ from Q. Thus if $P*_{i\gamma}$ is
obtained from $P_{i\gamma}$ by replacing all occurrences of
C, C' by $\Delta_\gamma, \Lambda_\gamma$ respectively ($i = e+1, \ldots, f$),
the formula

$$ {}_\delta \Gamma^f_{i=e+1} P*_{i\gamma} Q $$

will take the truth-value 1 whenever all
variable functors of Q take values corresponding
to the construction of $Q^{(\gamma)}$.

It then follows at once that if the formulae
$P*_{n+1}, \ldots, P*_e, P*_{e+1,3}, \ldots, P*_{f3}, \ldots, P*_{e+1,2^{\alpha(Q)}},$
$\ldots, P*_{f,2^{\alpha(Q)}}$ are re-named $S_1, \ldots, S_\eta$ respectively,
then the formula

$$ {}_\delta \Gamma^\eta_{i=1} S_i Q $$

always takes the truth-value 1 and is, by the
weak completeness, provable. If, in the
hypothetical deduction (using the classical
substitution rule)

$$ P_\lambda' \vdash P_{n+1}, \ldots, P_d, $$

we replace all occurrences of C by Δ we obtain,
in our present formalisation, the hypothetical
deduction

$$ P^\dagger_\lambda \vdash P*_{n+1}, \ldots, P*_d, $$

where $P^\dagger_\lambda$ is obtained from P_λ (using R') by
replacing all occurrences of variable functors by
Δ. Hence

$$ P_\lambda \vdash P*_{d+1}, \ldots, P*_d. $$

Similarly

$$ P_\mu \vdash P*_{d+1}, \ldots, P*_e. $$

If, in the hypothetical deduction

$$ P_\gamma^{(K)} \vdash P_{e+1,\gamma}, \ldots, P_{f\gamma}, $$

we replace all occurrences of C, C' by $\Delta_\gamma, \Lambda_\gamma$
respectively, we obtain, in our present
formalisation, the hypothetical deduction

$$ P^\dagger_{\gamma\gamma} \vdash P*_{e+1,\gamma}, \ldots, P*_{f\gamma}, $$

where $P^\dagger_{\gamma\gamma}$ is obtained from P_γ (using R') by
replacing those variable functors which were
replaced by C in the construction of $Q^{(\gamma)}$ by Δ_γ
and the others by Λ_γ. Thus

$$ P_\gamma \vdash P*_{e+1,\gamma}, \ldots, P*_{f\gamma} \quad (\gamma = 3, 4, \ldots, 2^{\alpha(Q)}). $$

Hence

$$ P_\lambda, P_\mu, P_\gamma \vdash S_1, \ldots, S_\eta $$

and, since

$$ \vdash {}_\delta \Gamma^\eta_{i=1} S_i, $$

$$ P_1, \ldots, P_k \vdash Q. $$

__Proof of the Main Theorem.__ We first re-name the
formulae $U_{m-1}, \ldots, U_1, V_{n-1}, \ldots, V_1, X_{n-1}, \ldots, X_1$
by $V_1, \ldots, V_{3n-3}$ respectively. It follows at once
that, if A_i, B_i, C_i correspond to the set
$\{V_1, \ldots, V_i\}$ ($i = 1, \ldots, 3m-3$) then not $(A_i, B_i, C_i) \leqslant$
$\equiv(V_{i+1})$ ($i = 1, \ldots, 3m-4$). Thus, by the lemma,
not $V_1, \ldots, V_i \vdash V_{i+1}$ ($i = 0, \ldots, 3n-4$)
(since, by plausibility, not $\vdash V_1$) and, for all Q,

$$ X_1, V_1 \vdash Q. $$

The degree of completeness is therefore at least
$3m-4$. Since the number of ordered triples $\equiv(Q)$ in
a sequence ordered in such a way that $\equiv(P)$
precedes $\equiv(Q)$ whenever $\equiv(Q) \leqslant \equiv(P)$ is at most
$3m-3$ the theorem now follows from the sufficiency
result of the lemma. $((m, m, m)$ is, of course,
ignored, as are triples (A, B, C) for which $A < C$ or $B < C$.)

A strictly analogous argument, using general
(cf. [5]) and the eliminability of negation
(cf. [7]), proves that, in the $\aleph_0$-valued case,
the degree of completeness is ω. Corresponding
arguments may also be used in cases related to the
set $\{F_1, \ldots, F_n\}$, where, in the 2^n-1 related
propositional calculi with constant functors
forming non-empty subsets of the given set, the
degree of completeness can be evaluated by the
general method given later in this paper for
finite-valued propositional calculi or by similar
methods (cf. [6]) in the $\aleph_0$-valued case. Strictly

analagous arguments show that, if the set possesses generalised functional completeness [4, 9] and the 2^a-1 related propositional calculi have degrees of completeness $D_1, ..., D_{2^a-1}$, all of which are finite, then the present system has degree of completeness

$$2-2^a + \sum_{i=1}^{2^a-1} D_i.$$

If any of the numbers $D_1, ..., D_{2^a-1}$ are infinite then the addition of these ordinals may not be commutative, and the degree of completeness will, by corresponding arguments, be the greatest value (as the ordering of the 2^a-1 calculi varies) of the least ordinal α for which

$$\alpha > \sum_{i=1}^{2^a-1} E_i,$$

whenever $E_i < D_i$ ($i=1, ..., 2^a-1$), subject to the restriction that if E_j, E_k correspond to subsets $\mathcal{E}_j$, $\mathcal{E}_k$ of the set $\{F_1, ..., F_a\}$ and $\mathcal{E}_j \subset \mathcal{E}_k$ then $k < j$.

As an example of the applicability of the above results we shall consider the case where $m \equiv 1 \pmod 3$, $SPQ = CPIQ$, $n=3$ and $\{F_1, F_2, F_3\} = \{c, c', s\}$. The functor S is a quasi Sheffer function for the m-valued Łukasiewicz propositional calculus [10] and generalised functional completeness holds [11] for the above set. The degrees of completeness corresponding to the subsets $\{c\}$, $\{c'\}$, $\{s\}$, $\{c, c'\}$, $\{c, s\}$, $\{c', s\}$, $\{c, c', s\}$ may then be evaluated in the required way (cf. [5] and [8]) as m, m, $1+d(m-1)$, m, $1+d(m-1)$, $1+d(m-1)$, $1+d(m-1)$ respectively, so the required degree of completeness is $3m+4d(m-1)-2$.

The general applicability of the result would depend on the construction of an algorithm for the evaluation of the degree of completeness of a wide class of propositional calculi. This problem was raised by Rosser and Turquette [14] for a wide class of formalisations which they gave (though it would, of course, be necessary to replace their axiom schemes by the corresponding axioms and adjoin the usual substitution rule) and we shall consider a closely related class of formalisations. We shall make the following assumptions:

(i) The formalisation is weakly complete and plausible and relates to an m-valued propositional calculus with s designated truth-values ($1 \leq s \leq m-1$, $2 \leq m < \aleph_0$).

(ii) In terms of the primitive functors, which we shall denote by $F_1, ..., F_a$ of $n_1, ..., n_a$ arguments respectively, we may define an implication functor I which satisfies the standard conditions of Rosser and Turquette [13]. (This condition is automatically satisfied if we

can define the implication functor of Łukasiewicz, in the case where $s=1$.)

(iii) The primitive rules of procedure are the rule of substitution for propositional variables and the rule of modus ponens with respect to I. (We shall use the operator Γ to denote summation by I with association to the right.)

We shall say that a subset $\mathcal{E}$ of $\{1, ..., m\}$ is closed if the truth-values of the formulae $F_i p_1 ... p_{n_i}$ ($i=1, ..., b$) belong to $\mathcal{E}$ whenever the truth-values of $p_1, ..., p_h$ ($h=\max(n_1, ..., n_b)$) belong to $\mathcal{E}$. Clearly there is an effective method of deciding whether a given subset is closed, so we may enumerate the closed subsets $\mathcal{E}_1, ..., \mathcal{E}_N$. We note that $2 \leq N \leq 2^m$. The N subsets are, of course, partially ordered by inclusion and we shall extend this, in future, to a simple ordering in which $\mathcal{E}_i \prec \mathcal{E}_j$ whenever $i < j$ and whenever $\mathcal{E}_i \subset \mathcal{E}_j$ ($i, j \in \{1, ..., n\}$). We shall say that a formula P is of class i if there is an assignment of truth-values in $\mathcal{E}_i$ to its propositional variables under which P takes an undesignated truth-value, but there is no such assignment in any of $\mathcal{E}_1, ..., \mathcal{E}_{i-1}$ which is a subset of $\mathcal{E}_i$. Thus, for example, if $m=5$, $N=9$, $\mathcal{E}_1 = \phi$, $\mathcal{E}_2 = \{1, 5\}$, $\mathcal{E}_3 = \{1, 2, 5\}$, $\mathcal{E}_4 = \{1, 3, 5\}$, $\mathcal{E}_5 = \{1, 4, 5\}$, $\mathcal{E}_6 = \{1, 2, 3, 5\}$, $\mathcal{E}_7 = \{1, 2, 4, 5\}$, $\mathcal{E}_8 = \{1, 3, 4, 5\}$, $\mathcal{E}_9 = \{1, ..., 5\}$ and P takes undesignated truth-values when its two propositional variables take the truth-values 2, 3 respectively or 2, 4 respectively, but not in the remaining 23 cases, then P is of classes 6, 7 but not of classes 1, 2, 3, 4, 5, 8, 9.

Lemma 1. If $P_1, ..., P_A$; Q are of classes $i_1, ..., i_A$; $j_1, ..., j_\beta$ (only) respectively and $\mathcal{E}_{i_\gamma} \subseteq \mathcal{E}_{j_c}$ for some d ($=d(c)$) $\in \{1, ..., \alpha\}$ ($c=1, ...; \beta$) then $P_1, ..., P_A \vdash Q$.

Let us denote the propositional variables of P_c, Q by $P_{1c}, ..., P_{a_c c}$; $Q_1, ..., Q_k$ respectively ($C=1, ..., A$). Let Q take an undesignated truth-value when $Q_1, ..., Q_k$ take the truth-values $y_{1\theta}, ..., y_{k\theta}$ respectively ($\theta=1, ..., h$) but not otherwise. Let the assignment $y_{1\theta}, ..., y_{k\theta}$ correspond to $\mathcal{E}_\tau$ ($1 \leq \tau \leq N$) but not to any proper subset $\mathcal{E}_i$ ($1 \leq i \leq N$) of $\mathcal{E}_\tau$ and let c be an integer such that $\mathcal{E}_{i_c} \subseteq \mathcal{E}_\tau$ ($c=c(\theta)$) and let an assignment of truth-values to the variables $P_{1c}, ..., P_{a_c c}$ corresponding to d(c) be $x_{1\theta c}, ..., x_{a_c \theta c}$ (for a suitable value of C). By the minimality property of $\mathcal{E}_\tau$, we may construct a formula $\Phi_{y_{1\theta} ... y_{k\theta} e}(Q_1, ..., Q_k)$ which takes the truth-value e when $Q_1, ..., Q_k$ take the truth-values $y_{1\theta}, ..., y_{k\theta}$ respectively provided that $e \in \mathcal{E}_\tau$. (Otherwise there would be a closed subset $\mathcal{E}_\mu$ containing $y_{1\theta}, ..., y_{k\theta}$ but not

211

e and $\mathcal{E}_\mu \cap \mathcal{E}_\gamma$ would be a closed proper subset of $\mathcal{E}_\gamma$.) Since $x_{1\theta c}, \ldots, x_{n_c \theta c} \in \mathcal{E}_{i_j} \subseteq \mathcal{E}_{i_j} \subseteq \mathcal{E}_\gamma$ we may construct the formulae $\Phi_{y_{1\theta}} \ldots y_{n_\theta} \gamma_{y\theta}(Q_1, \ldots, Q_k)$ ($\gamma =1, \ldots, n_c$). If we substitute these for P_{1c}, $\ldots, P_{n_c c}$ respectively in P_c we infer a formula $\Psi_\theta(Q_1, \ldots, Q_k)$ which takes an undesignated truth-value when $Q_1, \ldots, Q_k$ take the truth-values $y_{1\theta}$, $\ldots, y_{k\theta}$ respectively ($\theta =1, \ldots, h$). Hence, by the weak completeness,

$$\vdash \Gamma_{\theta=1}^h \Psi_\theta(Q_1, \ldots, Q_k)Q.$$

Since we have established that

$$P \vdash \Psi_\theta(Q_1, \ldots, Q_k) \quad (\theta=1, \ldots, h)$$

the lemma now follows at once by h uses of <u>modus ponens</u>.

Lemma 2. <u>If $P_1, \ldots, P_n \vdash Q$ and Q is of class i then there exists an integer j such that $1 \leq j \leq n$, P_j is of class c_j and $\mathcal{E}_{c_j} \subseteq \mathcal{E}_i$.</u>

If no such integer j exists then $P_1, \ldots, P_n$ take designated truth-values whenever all their propositional variables take truth-values in $\mathcal{E}_i'$. It then follows easily, by strong induction on the length of the derivation of Q (<u>cf.</u> [8]) that the latter formula takes designated truth-values whenever all its propositional variables take truth-values in $\mathcal{E}_i'$. Thus Q is not of class i and we have a contradiction.

It follows at once from Lemmas 1 and 2 that the longest sequence of formulae $P_1, \ldots, P_n$ such that

$$\text{not } P_1, \ldots, P_{k-1} \vdash P_k \quad (k=1, \ldots, n) \text{ and}$$
$$\text{for all } Q; \ P_1, \ldots, P_n \vdash Q$$

is obtained by choosing P_k to be of class $N+1-k$ ($k=1, \ldots, N-1$) only, provided that these $N-1$ formulae all exist. In that case the degree of completeness is N and the existence requirement is met provided that, in terms of $F_1, \ldots, F_t$, we can define functors $J_1, \ldots, J_m$ satisfying the standard conditions of Rosser and Turquette. We may then (<u>cf.</u> [13]) define standard condition negation (N) and conjunction (K) functors, so we may choose P_k to be

$$NKJ_{a_{1k}}P_1 KJ_{a_{2k}}P_2 \ldots KJ_{a_{j_k-1}k}P_{j_k-1}J_{a_{j_k}k}P_{j_k},$$

where $\mathcal{E}_{N+1-k}=\{a_{1k}, \ldots, a_{j_k}\}$ ($k=1, \ldots, N-1$).

In general, however, the existence requirement will not be met. (For example, if the only primitive functor is the m-valued implication functor of Łukasiewicz, then no formula is of the class corresponding to $\{1\}$ or, if $m>2$, of the class corresponding to $\{1, m\}$ only.) We have, as a means of coping with this situation, an algorithm for deciding whether a formula of

classes $i_1, \ldots, i_\gamma$ (only) ($\{i_1, \ldots, i_\gamma\} \subseteq \{1, \ldots, N\}$) exists. (This is given by slight extensions of the methods of Kalicki (see [2] and [12]). We may therefore construct the (finite) set $\mathcal{H}$ of sets of classes. Each ordered subset of $\mathcal{H}$ will then induce a corresponding sequence of formulae $P_1, \ldots, P_n$ (for some finite n) which may, by Lemmas 1 and 2, be tested to ascertain whether it has the properties of the sequence of the previous paragraph. (At least one acceptable sequence exists since not $\vdash p$ and $p \vdash Q$.) The successor of the length of the longest acceptable sequence is the required degree of completeness.

REFERENCES

[1] J. Jones, "A formalisation of an $\aleph_0$-valued propositional calculus with variable functors", Z. für Math. Logik, vol. 28 (1982), pp. 505-510.

[2] J. Kalicki, "A test for the existence of tautologies in many-valued logics", J. Symbolic Logic, vol. 15 (1950), pp. 182-184.

[3] J. Łukasiewicz and A. Tarski, "Untersuchungen über den Aussagenkalkül", Comptes rendus (Warsaw), Classe III, vol. 23 (1930), pp. 30-50. (For an English translation see [16].)

[4] A. Rose, "Many-valued propositional calculi without constants", Proc. ISMVL, vol. 9 (1979), pp. 128-134.

[5] A. Rose, "Le degré de saturation du calcul propositionnel implicatif à m valeurs de Łukasiewicz", Comptes rendus (Paris), vol. 240 (1955), pp. 2280-2281.

[6] A. Rose, "The degree of completeness of the $\aleph_0$-valued Łukasiewicz propositional calculus", J. London Math. Soc., vol. 28 (1953), pp. 176-184.

[7] A. Rose, "Formalisation du calcul propositionnel implicatif à $\aleph_0$ valeurs de Łukasiewicz, Comptes rendus (Paris), vol. 243 (1956), pp. 1183-1185.

[8] A. Rose, "The degree of completeness of the m-valued Łukasiewicz propositional calculus", J. London Math. Soc., vol. 27 (1952), pp. 92-102. (See also correction and addendum, ibid., vol. 44 (1969), pp. 587-596.)

[9] A. Rose, "A generalisation of the concept of functional completeness and applications to modus ponens", Z. für Math. Logik, vol. 28 (1982), pp. 317-322.

[10] A. Rose, "Some generalised Sheffer functions", Proc. Cambridge Phil. Soc., vol. 48 (1952), pp. 369-373.

[11] A. Rose, "Generalised functional completeness in Łukasiewicz propositional calculi", (submitted to NDJFL).

[12] A. Rose, Review of a paper by N. M. Martin, J. Symbolic Logic, vol. 16 (1951), pp. 275-276.

[13] J. B. Rosser and A. R. Turquette, Many-valued logics, North Holland, 1952.

[14] J. B. Rosser and A. R. Turquette, "Axiom schemes for m-valued propositional calculi, J. Symbolic Logic, vol. 10 (1945), pp. 61-82.

[15] A. Tarski, "Über einige fundamentalen Begriffe der Metamathematik", Comptes rendus (Warsaw), Classe III, vol. 23 (1930), pp. 22-29. (For an English translation see [16].)

[16] A. Tarski, Logic, semantics, metamathematics, Oxford University Press, 1956.

THREE-VALUED LOGIC AND ITS APPLICATION TO THE QUERY LANGUAGE OF INCOMPLETE INFORMATION

by Akira Nakamura

Department of Applied Mathematics, Hiroshima University
Higashi-Hiroshima, 724 Japan

ABSTRACT

Lipski has considered a mathematical model of incomplete information and discussed some problems related to it. Also, we introduced a second-order predicate logic corresponding to the query language and solved some interesting problems about the decidability of this language. This paper proposes a 3-valued ($\{1,1/2,0\}$) logic based on this model instead of the above second-order predicate logic. By the aid of this 3-valued logic we give an axiomatic system of this query language.

§0 Introduction

In [1], Lipski proposed a mathematical model of incomplete information and discussed some problems related to it. According to his proposal, proposition which express queries to an information storage and retrieval system can be regarded as a special kind of formulas of the first-order predicate logic. So, in [2] he gave two ways (i.e., *external* and *internal*) of interpreting formulas of the predicate logic, by making use of models of incomplete information. In regard to this interpretation, some similarities to Kripke models for modal logic are known. In fact, some relationships to modal logic S4 were mentioned in [2].

In [3], we introduced a second-order predicate logic corresponding to this query language and solved some interesting problems about the decidability of this language. In this paper, we propose a 3-valued ($\{1,1/2,0\}$) logic based on this model instead of the above second-order predicate logic. Further, by the aid of this 3-valued logic we give an axiomatic system of this query language. That is, we give a translation of formulas of the query language, called *extended formulas*, into formulas of this 3-valued logic and show that an extended formula is true in every interpretation if and only if the corresponding formula of this 3-valued logic is valid. Then, by making use of this result we give a complete tableau method for the extended formulas. The problem of this axiomatization for the query language has been an open question proposed in [2].

The key ideas used to show results of this paper are as follows:
(1) Incomplete information corresponds to the value 1/2 of the 3-valued logic.
(2) Formula of modal logic can be induced into the first-order predicate logic by introducing a new sort of domain. This corresponds to the fact: For an element w of the new domain, it is possible to give an interpretation of extended formula *at the world* w.

§1 Preliminaries

In this paper, we will use almost the same terminology and notations as in [2]. First, we will give a brief account of *internal interpretations* for a query language.

A first-order language L is a language which consists of a list of countable n-ary predicate symbols $P^n, Q^n, \ldots, P_1^n, P_2^n, \ldots$, for each $n \geq 1$, a list of countable individual variables $x, y, \ldots, x_1, x_2, \ldots$ the logical connectives $\neg, \wedge$, and the quantifier $\forall$. Other connectives $\vee, \supset, \equiv, \exists$ can be defined as abbreviations in the usual way. We suppose that L does not contain any function symbol and any individual or predicate constants. Then n denoting n-ary of predicate symbol is sometimes omitted. *First-order formulas* of L are defined in the usual way. Next, we add a unary connective $\square$ to L. The language thus obtained is denoted by L*. (First-order) formulas of L* are called *extended formulas*. In the following, formulas will be denoted by $\phi, \psi, \ldots$, or $\phi(x_1, \ldots, x_n), \psi(x_1, \ldots, x_n), \ldots$. (Some of variables $x_1, \ldots, x_n$ may not occur in $\phi(x_1, \ldots, x_n)$ and other variables may occur in it.)

Following Lipski [2], we will introduce internal interpretations of extended formulas.

Definition 1.1 An *incomplete model* (or a *model* for short) is a triple M=<X,u,U>, where X is a non-empty set called the individual domain of M, and u and U are mappings which associate some subsets $u(P) \subseteq U(P) \subseteq X^n$ for every n-ary predicate symbol P ($n \geq 1$).

If u=U holds in a model M=<X,u,U>, then M is said to be *complete*. Complete models are nothing but ordinary models for the first-order formulas, as explained later.

Definition 1.2 Given two models M_1=<X,u_1,U_1> and M_2=<X,u_2,U_2> with the same individual domain X, M_2 is an *extension* of M_1 ($M_1 \leqslant M_2$ or $M_2 \geqslant M_1$, in symbol) if and only if for every predicate symbol P $u_1(P) \subseteq u_2(P) \subseteq U_2(P) \subseteq U_1(P)$.

214

Let $\phi(x_1,\ldots,x_n)$ be any extended formula with free individual variables $x_1,\ldots,x_n$. For any model $M=<X,u,U>$ and $a_1,\ldots,a_n \in X$, we want to define the notation "$\phi(x_1,\ldots,x_n)$ *is satisfied in M when x_1, $\ldots,x_n$ are interpreted as $a_1,\ldots,a_n$, respectively*", in symbol

$$M \models \phi(a_1,\ldots,a_n).$$

To do so, we first extend our language L* by adding a new individual constant $\bar{a}$ for each $a \in X$. (By abuse of symbol, we will use the same letter a for $\bar{a}$, in the following.) The language thus obtained is denoted by L*[M].

Definition 1.3 Let $M=<X,u,U>$ be any model. For each closed extended formula ϕ of L*[M], define $M \models \phi$ recursively as follows:
1) $M \models P(a_1,\ldots,a_n)$ iff $(a_1,\ldots,a_n) \in u(P)$,

   where P is an n-ary predicate symbol,
2) $M \models \daleth \psi$ iff not $M \models \psi$,
3) $M \models \psi \wedge \theta$ iff $M \models \psi$ and $M \models \theta$,
4) $M \models \forall x \psi(x)$ iff for every $a \in X$ $M \models \psi(a)$,
5) $M \models \Box \psi$ iff for every $M' \geqslant M$ $M' \models \psi$.
   Next, let $\phi(x_1,\ldots,x_n)$ be any extended formula of L* with free individual variables $x_1,\ldots,x_n$.

Then, define
$$M \models \phi(x_1,\ldots,x_n) \quad \text{iff } M \models \forall x_1 \ldots \forall x_n \phi(x_1,\ldots,x_n).$$
Notice that $\forall x_1 \ldots \forall x_n \phi(x_1,\ldots,x_n)$ is a closed formula of L*[M], in the above definition. When M is a complete model, the definition of $M \models \phi$ coincides with the ordinary one, for every first-order formula ϕ.

Definition 1.4 Let ϕ and ψ be arbitrary extended formulas.
1) ϕ is *internally valid* if and only if $M \models \phi$ holds for every model M,
2) ϕ and ψ are *internally equivalent* if and only if $\phi \equiv \psi$ is internally valid.
   Here, we give the meaning of some notations.
$\forall x_1 \ldots \forall x_n (P(x_1,\ldots,x_n) \supset Q(x_1,\ldots,x_n))$ is denoted by $P \leqslant Q$. Also, $P \leqslant Q \wedge Q \leqslant R$ is represented by $P \leqslant Q \leqslant R$.

§2 Three-valued predicate logic $_3L$

We consider a 3-valued predicate logic $_3L$. The symbols of this logic are the same as in the usual predicate logic except the logical symbols and the special individual symbols. Further, well-formed formulas (wff's) are defined in the following way:
(1) The arguments of every n-ary predicate symbol $P(.,\ldots,.)$ must be occupied by the special individual symbol w_i ($0 \leq i$) at the n-th argument.

(2) Other construction rules except (1) are the same as in the usual one.
   Logical symbols are $\wedge, \vee, \supset, \daleth, J_1, J_{1/2}, J_0$. $\wedge$, $\vee$, and $\supset$ are duadic and $J_1, J_{1/2}, J_0, \daleth$ are monadic.
$\forall x_1 \ldots \forall x_n (P(x_1,\ldots,x_n,w_i) \supset Q(x_1,\ldots,x_n,w_j))$ is denoted by $P(x_1,\ldots,x_n,w_i) \leqslant Q(x_1,\ldots,x_n,w_j)$ in the similar way to L*.
   The semantics of this logic is defined as follows:

First, the truth values are 1,1/2,0 and they mean *truth, unknown, false*, respectively. Domains of individual constants have two sorts X and W. X is the usual domain and W is the special domain, called the set of *worlds*. The ordinary individual variable ranges over X, but the special individual variable w_i over W. Let $\mathcal{C}$ be a closed wff of $_3L$.
Then, $v(\mathcal{C})$ stands for a valuation v of $\mathcal{C}$. The truth value function of $\vee, \wedge, \supset, \daleth, J_1, J_{1/2}, J_0$ are

defined as follows:
  $v(\mathcal{C} \vee \mathcal{L}) = \max(v(\mathcal{C}), v(\mathcal{L}))$,
  $v(\mathcal{C} \wedge \mathcal{L}) = \min(v(\mathcal{C}), v(\mathcal{L}))$,
  $v(\mathcal{C} \supset \mathcal{L}) = \min(1, v(\mathcal{L}) - v(\mathcal{C}) + 1)$,
  $v(\daleth \mathcal{C}) = 1 - v(\mathcal{C})$,
  $v(J_i \mathcal{C}) = 1$ if $v(\mathcal{C}) = i$
         $= 0$ otherwise.
Further, we use in the usual way the following definition for closed wff's $\forall x \mathcal{C}$ and $\forall w_i \mathcal{C}$. That is as follows:
  Let x be an ordinary individual variable.
  $v(\forall x \mathcal{C}(x)) = 1$ if for all $a \in X$ $v(\mathcal{C}(a)) = 1$,
           $= 1/2$ if for all $a \in X$ $v(\mathcal{C}(a)) \geq 1/2$ and
                    there exists $b \in X$ such that
                    $v(\mathcal{C}(b)) = 1/2$.
           $= 0$ if there exists $a \in X$ such that
                    $v(\mathcal{C}(a)) = 0$.
  Let w_i be a special individual variable.
  $v(\forall w_i \mathcal{C}(w_i)) = 1$ if for all $r \in W$, $v(\mathcal{C}(r)) = 1$,
           $= 1/2$ if for all $r \in W$, $v(\mathcal{C}(r)) \geq 1/2$
                and there exists $s \in W$ such
                that $v(\mathcal{C}(s)) = 1/2$,
           $= 0$ if there exists $r \in W$ such that
                $v(\mathcal{C}(r)) = 0$.
$\exists x \mathcal{C}$ and $\exists w_i \mathcal{C}$ are defined in the similar way.

The validity and satisfiability of wff $\mathcal{C}$ of $_3L$ are also defined in the usual way.

§3 Embedding theorem

Let S_1 and S_2 be the sets of extended wff's and $_3L$, respectively. Also, define a transformation τ on the set S_2 as follows:
  $\tau(\mathcal{C})$ is a formula obtained from $\mathcal{C}$ by replacing (irrespectively of *free* or *bounded*) $P(x_1,\ldots,x_n,w_i)$ by $P(x_1,\ldots,x_n,w_{i+1})$ for each P and each i of $\mathcal{C}$.

Then, we define a mapping $f: S_1 \to S_2$ as follows:
(1) If $\phi = P(x_1,\ldots,x_n)$ then $f(\phi) = J_1 P(x_1,\ldots,x_n,w_0)$,

(2) If $\phi = \daleth \psi$ then $f(\phi) = \daleth f(\psi)$,
(3) If $\phi = \psi \wedge \theta$ then $f(\phi) = f(\psi) \wedge f(\theta)$,
(4) If $\phi = \forall x \psi$ then $f(\phi) = \forall x f(\psi)$,
(5) Let $\phi = \Box \psi$ and $P_1,\ldots,P_k$ be all predicate symbols appearing in ϕ. Then,
$$f(\phi) = \forall w_1 ((J_1 P(x_1,\ldots,x_n,w_0) \leqslant J_1 P(x_1,\ldots,x_n,w_1))$$
$$\wedge (J_1 P(x_1,\ldots,x_n,w_1) \vee J_{1/2} P(x_1,\ldots,x_n,w_1)$$
$$\leqslant J_1 P(x_1,\ldots,x_n,w_0) \vee J_{1/2} P(x_1,\ldots,x_n,w_0))$$
$$\supset \tau(f(\psi))),$$
where $J_1 P(x_1,\ldots,x_n,w_0) \leqslant J_1 P(x_1,\ldots,x_n,w_1)$
means

215

$(J_1P_1(x_1,\ldots,x_n,w_0) \leqslant J_1P_1(x_1,\ldots,x_n,w_1))_\wedge\cdots_\wedge$
$(J_1P_k(x_1,\ldots,x_n,w_0) \leqslant J_1P_k(x_1,\ldots,x_n,w_1))$ and

$J_1P(x_1,\ldots,x_n,w_1)\vee J_{1/2}P(x_1,\ldots,x_n,w_1) \leqslant$
$\quad J_1P(x_1,\ldots,x_n,w_0)\vee J_{1/2}P(x_1,\ldots,x_n,w_0)$ has the similar meaning.

__Lemma 3.1__ Let $M=<X,u,U>$ and $_3M=<X^\cup W,v,v>$ be models for L^* and $_3L$, respectively. Let $\bar{q}_m$ be a constant such that the corresponding q_m is in W. Moreover, suppose that for every predicate symbol P of L^* $u(P)=\{(\ldots)|v(J_1P(\ldots,q_m))\}$ and $U(P)=\{(\ldots)|$ $v(J_1P(\ldots,q_m)\vee J_{1/2}P(\ldots,q_m))\}$ hold. Then, for any closed extended formula ϕ of $L^*[M]$

$$M \models \phi \qquad \text{iff} \quad _3M \models \tau^m(f(\phi(q_0))),$$

where $\tau^m(f(\phi(q_0)))$ means a formula obtained from $\tau^m(f(\phi))$ by substituting q_m for a special individual variable w_m.

Proof

We will show this lemma for every M and $_3M$ and m, by the induction on number of logical connectives in ϕ.

(1) The case where ϕ is $P(a_1,\ldots,a_n)$ for some $a_1,$ $\ldots, a_n\in X$.

Then, $\tau^m(f(\phi(q_0)))=J_1P(a_1,\ldots,a_n,q_m)$.

So, $M\models P(a_1,\ldots,a_n)$ iff $(a_1,\ldots,a_n)\in u(P)=$
$$v(J_1P(a_1,\ldots,a_n, q_m))$$
$$\text{iff } _3M\models\tau^m(f(P(a_1,\ldots,a_n,q_0))).$$

(2) Induction step

We will prove this lemma only for the case where ϕ is of the form $\Box\,\psi$. Other cases can be provable easily. For the sake of brevity, we suppose that predicate symbols appearing in ϕ are only P and that P is k-ary.

By the definition,
$M \models \Box\psi$ iff for every $M'\geqslant M$ $M'\models\psi$.
So, it is sufficient to show that
(3.1) for every $M'\geqslant M$ $M'\models\psi$
if and only if
(3.2) $_3M \models \tau^m(f(\phi(q_0)))$.

We remark here that
$\tau^m(f(\phi(q_0)))$
$= \forall w_{m+1}((J_1P(x_1,\ldots,x_k,q_m) \leqslant J_1P(x_1,\ldots,x_k,w_{m+1}))_\wedge$
$\quad(J_1P(x_1,\ldots,x_k,w_{m+1})\vee J_{1/2}P(x_1,\ldots,x_k,w_{m+1}) \leqslant$
$\quad J_1P(x_1,\ldots,x_k,q_m)\vee J_{1/2}P(x_1,\ldots,x_k,q_m))$
$\quad\supset\tau^{m+1}(f(\psi)))$.

We consider $J_1P(x_1,\ldots,x_k,q_m)$, $J_1(x_1,\ldots,x_k,q_m)\vee$ $J_{1/2}P(x_1,\ldots,x_k,q_m)$ as the corresponding formulas of P^m_*, P^{m*} in [3], respectively.

First, let us assume that (3.1) holds. For an arbitrary constant q_{m+1}, let $A=J_1P(x_1,\ldots,x_k,q_{m+1})$ and $B=J_1P(x_1,\ldots,x_k,q_{m+1})\vee J_{1/2}P(x_1,\ldots,x_k,q_{m+1})$ such that

(3.3) $_3M\models J_1P(x_1,\ldots,x_k,q_m) \leqslant A \leqslant B$
$$\leqslant J_1P(x_1,\ldots,x_k,q_m)\vee J_{1/2}P(x_1,\ldots,x_k,q_m).$$

Now, let us define a model $M'=<X,u',U'>$ by $u'(P)$ $=v(A)$ and $U'(P)=v(B)$. Since $M\leqslant M'$ holds, $M'\models\psi$. By the hypothesis of induction, we have
(3.4) $_3M \models \tau^{m+1}(f(\psi(q_0)))^+$,

where $\tau^{m+1}(f(\psi(q_0)))^+$ denotes the formula obtained from $\tau^{m+1}(f(\psi))$ by replacing each occurence of $J_1P(x_1,\ldots,x_k,w_{m+1})$ and $J_1P(x_1,\ldots,x_k,w_{m+1})\vee$ $J_{1/2}P(x_1,\ldots,x_k,w_{m+1})$ by predicate constants A and B, respectively. Since (3.4) holds from (3.3) for each A and B, (3.2) holds.

Conversely, suppose that (3.2)holds. Let $M'=<$ $X,u',U'>$ be any model such that $M\leqslant M'$. Define subsets A and B of X^k by $u'(P)=A$ and $U'(P)=B$, respectively. In this case by making use of a constant q_{m+1} in W and v of $_3M$, A and B are always representable by $\{(x_1,\ldots,x_k)|v(J_1P(x_1,\ldots,x_k,q_{m+1}))\}$ and $\{(x_1,\ldots,x_k)|v(J_1P(x_1,\ldots,x_k,q_{m+1})\vee J_{1/2}P(x_1,$ $\ldots,x_k,q_{m+1}))\}$, respectively.

Then, it holds that for the above $_3M$
$_3M \models J_1P(x_1,\ldots,x_k,q_m) \leqslant A \leqslant B \leqslant$
$\quad J_1P(x_1,\ldots,x_k,q_m)\vee J_{1/2}P(x_1,\ldots,x_k,q_m)$
because $M\leqslant M'$. So, $_3M \models \tau^{m+1}(f(\psi(q_0)))^+$ holds too, where $\tau^{m+1}(f(\psi(q_0)))^+$ is the formula defined above. By the hypothesis of induction, $M'\models\psi$.
Thus, $M\models\phi$. //

__Theorem 3.2__ For any extended formula ϕ of L^*, ϕ is internally valid if and only if $f(\phi)$ is valid in $_3L$.

Proof

Let $P_1,\ldots,P_h$ be all predicate symbols appearing in ϕ. Suppose that ϕ is not internally valid. Then, there exists a model $M=<X,u,U>$ such that $M\models$ ϕ does not hold. Let $_3M=<X^\cup W,v,v>$ be a model such that $v(J_1P_i(x_1,\ldots,x_{i_k},q_0))=u(P_i)$ and $v(J_1P_i$ $(x_1,\ldots,x_{i_k},q_0)\vee J_{1/2}P_i(x_1,\ldots,x_{i_k},q_0))=U(P_i)$ for every $i=1,\ldots,h$, where q_0 is a constant in W. Then, it is obvious that $u(P_i)\subseteq U(P_i)$ for all $i=1,\ldots,h$. Therefore, $_3M \models f(\phi)$ does not hold by Lemma 3.1 . Hence, $f(\phi)$ does not valid in $_3L$.

Conversely, suppose that $_3M'\models f(\phi)$ does not hold for some model $_3M'=<X'^\cup W',v',v'>$.Define $M'=<X',u',U'>$ by $u'(P_i)=v'(J_1P_i(x_1,\ldots,x_{i_k},q_0))$ and $U'(P_i)=v'(J_1$ $P_i(x_1,\ldots,x_{i_k},q_0)\vee J_{1/2}P_i(x_1,\ldots,x_{i_k},q_0))$ for $i=1,$ $\ldots, h$, where q_0 is a constant in W'. Then, M' is a really incomplete model. Moreover, by Lemma 3.1 $M'\models\phi$ does not hold since $_3M'\models f(\phi)$ does not hold. Then, ϕ is not internally valid. //

Theorem 3.2 is considered as an embedding theorem of wff's in L^* into $_3L$.

216

§4 Method of tableaux for L*

In this section, we show a method of tableaux for L* and prove its completeness. This method is obtained by the aid of Theorem 3.2 from the usual one for the first-order predicate logic which is explained in [4]. We use here the same terminology as in [4]. That is, we use a *complete branch*, a *closed tableau*, an *open tableau*, an *open branch* and so on. The usual analytic tableau method for the first-order predicate logic is as follows:

Let ϕ and ψ be wff's.

1)
$$\frac{T \neg \phi}{F \phi} \qquad \frac{F \neg \phi}{T \phi}$$

2)
$$\frac{T(\phi \wedge \psi)}{T \phi \ , T \psi} \qquad \frac{F(\phi \wedge \psi)}{F \phi \mid F \psi}$$

3)
$$\frac{T(\phi \vee \psi)}{T \phi \mid T \psi} \qquad \frac{F(\phi \vee \psi)}{F \phi \ , F \psi}$$

4)
$$\frac{T(\phi \supset \psi)}{F \phi \mid T \psi} \qquad \frac{F(\phi \supset \psi)}{T \phi \ , F \psi}$$

5)
$$\frac{T \forall x \phi}{T \phi|_a^x} \qquad \frac{F \exists x \phi}{F \phi|_a^x}$$

where a is any individual,

6)
$$\frac{T \exists x}{T \phi|_a^x} \qquad \frac{F \forall x}{F \phi|_a^x}$$

where a is a new individual.

Here, we modify the above usual method to L*. In the following, we use the symbols $\mathbb{P}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{H}$ which correspond respectively to one in (5) defining the mapping $f: S_1 \rightarrow S_2$. Now, the method is as follows:

The rules 1)-6) are the same as above, but the following rules 7)-10) are defined.

7)
$$\frac{T \Box \phi}{T(\ \mathbb{P} \leqslant \mathbb{Q} \leqslant \mathbb{R} \ (\leqslant \mathbb{H}) \supset \rho(\phi))}$$

8)
$$\frac{F \Diamond \phi}{T(\ \mathbb{P} \leqslant \mathbb{Q} \leqslant \mathbb{R} \ (\leqslant \mathbb{H}) \supset \neg \rho(\phi)).}$$

In the above 7), $(\leqslant \mathbb{H})$ does not occur in the case that there exists no $\mathbb{P} \leqslant \mathbb{H}$ in the upward path from $T \Box \phi$. But, otherwise $(\leqslant \mathbb{H})$ is really $\leqslant \mathbb{H}$. $(\leqslant \mathbb{H})$ of 8) has the same meaning. Further, $\mathbb{P}$ appears in ϕ, but $\mathbb{Q}$, $\mathbb{R}$ are *arbitrary* predicates. Furthermore, $\rho(\phi)$ means that every predicate $\mathbb{P}$ in ϕ is replaced by $\mathbb{Q}$.

9)
$$\frac{T \Diamond \phi}{T(\ \mathbb{P} \leqslant \mathbb{Q} \leqslant \mathbb{R} \ (\leqslant \mathbb{H})), \quad T \rho(\phi)}$$

10)
$$\frac{F \ \phi}{T(\ \mathbb{P} \leqslant \mathbb{Q} \leqslant \mathbb{R} \ (\leqslant \mathbb{H})), \quad F \rho(\phi) .}$$

In 9) and 10), $(\leqslant \mathbb{H})$ has the same meaning mentioned above. $\mathbb{P}$ appears in ϕ, but $\mathbb{Q}$, $\mathbb{R}$ are *new* predicates. $\rho(\phi)$ is the same to 7).

Predicate $\mathbb{Q}$ introduced in the rules 7)-10) are called *extended predicates* of $\mathbb{P}$.

Then, a closed branch of this tableau method is defined as follows:

For an ordinary individual a, if TP(a) and FP(a) appear in one branch, then this branch is closed.

Here, to prove our main theorem we introduce a *modified Hintikka set* S (for an individual domain X) and show a lemma. A modified Hintikka set is a set S that the following conditions hold for every wff of L*.

(1) No prime formula and its negation are both in S.
(2) If $\phi \wedge \psi \in S$, then ϕ, ψ are both in S.
(3) If $\phi \vee \psi \in S$, then $\phi \in S$ or $\psi \in S$.
(4) If $\forall x \phi \in S$, then for every $k \in X$ $\phi(k) \in S$.
(5) If $\exists x \phi \in S$, then for at least one element $k \in X$ $\phi(k) \in S$.

Let $P_1, \ldots, P_k$ be all predicate symbols occuring in ϕ, then $\widetilde{\phi}$ be a formula obtained from ϕ by substituting their extended predicate symbols for $P_1, \ldots, P_k$.

(6) If $\Box \phi \in S$, then for every set of extended predicate symbols $\widetilde{\phi} \in S$.
(7) If $\Diamond \phi \in S$, then for at least one set of extended predicate symbols $\widetilde{\phi} \in S$.

Then, we have the following lemma:

Lemma 4.1 (Modified Hintikka's Lemma for L*) Every modified Hintikka set S for an individual domain X is satisfiable.

Proof

First, we consider a partial system $_3L^\circ$ of $_3L$ in which every prime formula has always J-connective in front of it. In $_3L^\circ$, we can easily prove Hintikka's Lemma in the usual way. Because, $J_i P(\ldots)$ is considerable as an ordinary predicate in the first-order predicate logic.

Now, we consider a correspondence of $P(\ldots)$ of $\Box \psi$ in L* to $J_1 P(\ldots, w_i)$ in the form $w_{i+1}(\phi_1 \supset \phi_2)$ of $f(\Box \psi)$. Thus, it is possible to give a correspondence of an extended predicate Q of P to a prime formula obtained from $J_1 P(\ldots, w_{i+1})$ by substituting arbitrary constant for this w_{i+1}. Similarly, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{H}$ in 7)-10) are considered, respectively, corresponding formulas $J_1 P(\ldots, w_{i+1})$, $J_1 P(\ldots, w_{i+1}) \vee J_{1/2} P(\ldots, w_{i+1})$, $J_1 P(\ldots, w_i) \vee J_{1/2} P(\ldots, w_i)$.

Then, it follows from the definition of $f: S_1 \rightarrow S_2$ and Theorem 3.2 that our rules 1)-10) are quite similar to the usual one. Thus, we get this lemma by the same technique as the case of the first-order predicate logic. //

Now, terms a *systematic tableau* and a *finished tableau* are defined in the similar way as in [4]. In this case, this systematic tableau is also defined for 7) and 8). Then, we have the following theorem.

Theorem 4.2 (Completeness Theorem of Modified Tableaux for L*) If ϕ is valid, then ϕ is provable — i.e., there exists a closed tableau for $F\phi$. Indeed, if ϕ is valid, then the systematic tableau for $F\phi$ must be closed after finite many steps.

Proof

    Suppose that ϕ is valid. Let T be the finished systematic tableau starting with Fϕ . If T contains an open branch B, then by Lemma 4.1 B would be satisfiable, hence Fϕ , being a term of B would be satisfiable, contrary to the hypothesis. Thus, ϕ is provable.

    Concerning the second statement, by König lemma, a closed infinite tableau is impossible, because if T is closed then every branch of T is finite length, hence T must be finite. //

    From Theorem 4.2, we are able to give a complete axiomatic system for L* in the familiar method.

§5 Conclusion

    We have proposed a 3-valued logic based on the query language of incomplete information. By the aid of this 3-valued logic we have given a complete axiomatic system of the query language. There would be further interesting research topics on applications of the 3-valued logic to theories of incomplete information.

References

[1] W. Lipski, Jr.: On semantic issues connected with incomplete information data bases, ACM Trans. on Databases Systems, 4.3 (1979) 262-296.

[2] W. Lipski, Jr.: On the logic of incomplete information, Proc. 6th International Symposium on Mathematical Foundations of Computer Science , Tatranska Lomnica, 1977, Lecture Notes in Computer Science 55, Springer-Verlag, Berlin (1977) 374-381.

[3] H. Ono and A. Nakamura: Decidability results on query language for data bases with incomplete information, Proc. 9th International Symposium on Mathematical Foundations of Computer Science, 1980, Lecture Notes in Computer Science 88, Springer-Verlag, Berlin (1980) 452-459.

[4] R. M. Smullyan: First-order Logic, Springer-Verlag, Berlin, Heiderberg, New York, 1968.

TOWARDS A FORMAL MULTI-VALUED UTILITY THEORY*

Michael Katz

School of Education, Haifa University, Haifa, Israel

ABSTRACT**

The paper provides an axiomatization in a
multi-valued language of some basic predicates of
utility theory. This axiomatization is based on
deductions interpreted so that assertions which are
theoretically error-free can be treated in an ap-
proximate way in actual structures for the language.

1. LANGUAGE AND LOGIC**

Utility theory, as well as other theories of
Social, Behavioral and Economic Sciences, is par-
ticularly suited to test the usefulness of multi-
valued approaches. These theories, in fact, are
based on notions like preference and indifference,
which empirically are strongly interconnected with
approximation and errors, while there is a need to
consider them as crisp notions at a general theo-
retical level.

To handle this we introduce what in [3] and
[4] we called "the logic of approximation" (which
is analyzed algebraically in our paper "Quotient
Algebras for Logics of Imprecision" in this volume).
This logic is based on a certain version of the
Łukasiewicz Logic, where truth values are considered
degrees of inexactness, and hence the usual seman-
tic rules are reversed, since the smaller the truth-
value, i.e., the degree of inexactness, the truer
the assertion (and see Giles [1] and Scott [9] for
similar ideas).

We start with a formal language L containing
variables $(u,v,w,u_1,u_2,...)$, predicate symbols $(p_m,
i,g,p,b,...)$ and the usual connectives $(\neg,\vee,\wedge,\rightarrow)$.
In a structure $\underline{X}$ for L with domain X every
m-place predicate p_m of L is interpreted by a
$[0,1]$-valued function $\underline{p}_m$ on X^m. If $\bar{x}$ is a func-
tion from a set containing all variables of a for-
mula φ of L into the domain X of $\underline{X}$ we denote
by $\varphi\bar{x}$ the truth-value of φ at $\bar{x}$. These truth-
values are defined as follows:

$$p_m(u_1,...,u_n)\bar{x} = \underline{p}_m(\bar{x}(u_1),...,\bar{x}(u_m))$$

$$\neg\psi\,\bar{x} = 1 - \psi\bar{x}$$

$$(\psi\vee\theta)\bar{x} = \min(\psi\bar{x},\theta\bar{x})$$

$$(\psi\wedge\theta)\bar{x} = \max(\psi\bar{x},\theta\bar{x})$$

$$(\psi\rightarrow\theta)\bar{x} = \max(0,\theta\bar{x} - \psi\bar{x}).$$

The special feature of the logic of approxima-
tion is the way it treats deductions. A deduction
of the form $\Gamma \vdash \Delta$ (read: "from Γ deduce Δ", where
Γ and Δ are finite sets of formulae of L) is to
hold in a structure for L if the error in at least
one member of Δ becomes as small as we wish when-
ever the errors in all members of Γ are made small
enough. More precisely, in the logic of approxima-
tion the structure $\underline{X}$ for L (with domain X) is
a model of the deduction $\Gamma \vdash \Delta$ if for every (posi-
tive real number) ε there is a (positive real
number) δ s.t. for all $\bar{x} \in X^{\{u_1,...,u_n\}}$

$$(\wedge\Gamma)\bar{x} < \delta \Rightarrow (\vee\Delta)\bar{x} < \varepsilon.$$

Here $\wedge\Gamma$ is the conjunction of all members of Γ,
$\vee\Delta$ is the disjunction of all members of Δ, $\Rightarrow$ de-
notes implication in the meta-language, and
$u_1,...,u_n$ are all the variables of members of $\Gamma\cup\Delta$.

2. THE AXIOM SYSTEM

Our elementary utility theory is based on
unary "gain" predicates, binary "indifference" and
"preference" predicates and ternary "betweenness"
predicates (g,i,p, and b, respectively). The for-
mula g(u) asserts that the object to which u
refers is "gainful" (or "utilizable"); i(u,v) as-
serts that the objects to which u and v refer
are "indifferent"; p(u,v) asserts that the object
to which u refers is "preferred" to the object to
which v refers (this is a non-strict preference,
admitting also indifference); b(u,v,w) asserts
that the object to which v refers is "between"
the objects to which u and w refer. (Gainful,
preferred, etc., only up to a certain error; hence
the inverted commas).

The following list of deduction expressions
(or "axioms") is our formal theory of utility in
the logic of approximation:

(1) $\vdash i(u,u)$

(2) $\vdash i(u,v) \rightarrow i(v,u)$

(3) $\vdash i(u,v) \rightarrow \big(i(v,w) \rightarrow i(u,w)\big)$

(4) $i(u,v) \vdash g(u) \rightarrow g(v)$

(5) $\vdash p(u,v),p(v,u)$

(6) $\vdash p(u,v) \rightarrow \big(p(v,w) \rightarrow p(u,w)\big)$

(7) $p(u,v),p(v,u) \vdash i(u,v)$

(8) $i(u,v) \vdash p(u,w) \rightarrow p(v,w)$

(9) $i(u,v) \vdash p(w,u) \rightarrow p(w,v)$

(10) $\vdash b(u,v,w),b(v,w,u),b(w,u,v)$

(11) $b(u,v,w) \vdash b(w,v,u)$

(12) $b(u,u_1,v),b(u,u_2,v) \vdash b(u,u_1,u_2),$
$b(u,u_2,u_1)$

(13) $b(u,v,u) \vdash i(u,v)$

(14) $\big(i(v,w) \rightarrow i(w,u)\big) \rightarrow i(u,v) \vdash b(u,v,w)$

(15) $b(u_1,u_2,v),b(u_2,u_3,v) \vdash b(u_1,u_2,u_3),$
$i(u_2,v) \wedge i(u_3,v)$

(16) $i(u,v) \vdash b(u,u_2,u_3) \rightarrow b(v,u_2,u_3)$

(17) $i(u,v) \vdash b(u_1,u,u_3) \rightarrow b(u_1,v,u_3)$

(18) $i(u,v) \vdash b(u_1,u_2,u) \rightarrow b(u_1,u_2,v)$.

All of these axioms have plausible intuitive meanings. We show this by looking at the following three groups of axioms.

(A) Axioms (1), (2) and (3) make indifference an equivalence relation in L. Yet in a structure $\underset{\sim}{X}$ for L they translate to a (pseudo-) metric on the domain X of $\underset{\sim}{X}$. In particular the transitivity axiom (3) translates to a triangle inequality in $\underset{\sim}{X}$. This seems an interesting and useful treatment of indifference as transtive in principle (as in, e.g., Luce and Raiffa [5], p.28) but intrasitive, or only partly transitive, due to measurement errors, in practice (as in, e.g., Roberts [6]).

(B) Axioms (5), (6) and (7) are connectedness, transitivity and anti-symmetry-up-to-indifference deductions for preference in L. They are satisfied in an approximate manner, according to our notion of deduction, in a model $\underset{\sim}{X}$ of the theory. For example, preference transitivity in L translates again to triangle inequality in X so that it holds fully in principle (as in, e.g., Luce and Raiffa [5], p.332) but not in practice (as in, e.g., Tversky [11]).

(C) Axioms (4), (8), (9), (16), (17) and (18) are substitutability properties in L w.r.t. i. In a model $\underset{\sim}{X}$ of our theory they translate to uniform continuity conditions w.r.t. i. For instance, (4) says that any two objects of $\underset{\sim}{X}$ which are nearly indifferent are about equally gainful (more precisely, the assertions that each of them is gainful are about equally erroneous). That is to say, for every ε there is a δ s.t. for every x and y in the domain X of $\underset{\sim}{X}$

$$\underset{\sim}{i}(x,y) < \delta \rightarrow |g(x) - g(y)| < \varepsilon .$$

In conclusion indifference and preference have

all the properties of classical equality and ordering (and their inter-relations) in the formal theory but they satisfy these properties only approximately in models of the theory. Gain predicates can be considered inverse utility functions in a model X: the higher the value of $g(x)$ in [0,1], i.e., the higher the error in asserting that the element x of the domain X of $\underset{\sim}{X}$ is gainful, the smaller the utility attached to x. Finally, betweenness, as axiomatized in (10) to (15), is analogous in our logic of approximation to Tarski's [10] "classical" betweenness and Roberts' [7] "tolerance" betweenness.

3. UNIDIMENSIONALITY

In this section an indifference structure is a structure $\underset{\sim}{X}$ for L which is a model of deductions (1), (2), and (3), for some binary predicate i of L. Given such a predicate i and a corresponding indifference structure $\underset{\sim}{X}$ we define the following notions w.r.t. these X and i.

(a) A unary predicate g of L is a gain predicate in $\underset{\sim}{X}$ if $\underset{\sim}{X}$ is a model of axiom (4) for this g. It is unidimensional in $\underset{\sim}{X}$ if in addition $\underset{\sim}{X}$ is a model of the deduction

$$g(u),g(v) \vdash i(i,v) .$$

(b) A binary predicate p of L is a preference predicate in $\underset{\sim}{X}$ if $\underset{\sim}{X}$ is a model of axioms (5) to (9) for this p. It is unidimensional in $\underset{\sim}{X}$ if there is a unary predicate g of L which represents p in $\underset{\sim}{X}$ in the sense that $\underset{\sim}{X}$ is in addition a model of the deductions

$$p(u,v) \vdash g(v) \rightarrow g(u)$$

$$g(v) \rightarrow g(u) \vdash p(u,v) .$$

(c) A ternary predicate b of L is a betweenness predicate in $\underset{\sim}{X}$ if $\underset{\sim}{X}$ is a model of axioms (10) to (18) for this b. It is unidimensional in $\underset{\sim}{X}$ if there is a binary predicate p of L which represents b in $\underset{\sim}{X}$ in the sense that $\underset{\sim}{X}$ is in addition a model of the deductions

$$p(u,v),p(v,w) \vdash b(u,v,w)$$

$$b(u,v,w) \vdash p(u,v) \wedge p(v,w),p(w,v) \wedge p(v,u).$$

In (a) we feel that it makes sense to say that if every two elements of X which are "almost fully" gainful are "practically" indifferent then gain cannot have several dimensions, or attributes, in X. In (b) and (c) the unidimensionality idea is that the objects of $\underset{\sim}{X}$ are ordered(by p or b) from the more to the less truely gainful (or vice versa). This, and the fact that gain is then itself unidimensional in the sense of (a), is the content of the following theorems (which are only a few of the interesting results which can easily be obtained w.r.t. unidimensionality).

(i) If the predicate p of L is a unidimensional preference predicate in the indifference structure $\underset{\sim}{X}$ then the predicate g of (b) above is a unidimensional gain predicate in $\underset{\sim}{X}$.

220

(ii) If the predicate b of L is a unidimensional betweenness predicate in the indifference structure $\underline{X}$ and the predicate p of (c) above is transitive in $\underline{X}$, i.e., $\underline{X}$ is a model of axiom (6) for this p, then p is a preference predicate in $\underline{X}$.

(iii) If b, p and $\underline{X}$ are as in (ii), and in addition p is unidimensional in $\underline{X}$, then the predicate g of (b) above represents b in $\underline{X}$ in the sense that $\underline{X}$ is a model of the deductions

$$g(u) \to g(v), g(v) \to g(w) \vdash b(u,v,w)$$

$$b(u,v,w) \vdash \big(g(u) \to g(v)\big) \wedge \big(g(v) \to g(w)\big),$$

$$\big(g(w) \to g(v)\big) \wedge \big(g(v) \to g(u)\big).$$

It should be clear that instead of defining unidimensionality for preference we can define, in a similar way, modifying (b) above, unidimensionality of indifference (or derive any one of these two from the other one). Then, modifying also (c), we can prove theorems like (i), (ii) and (iii) with indifference replacing preference. So we have "representations" of indifference, preference and betweenness by gain predicates.

It is interesting to note that these representations can be strenthened as follows. We add certain axioms to the formal theory (see, e.g., our [2] or [3]), to guarantee that in a model $\underline{X}$ of the extended theory one can build a function g, with the properties of the interpretation of a unidimensional gain predicate in $\underline{X}$, which satisfies for the interpretations of i, p, b in $\underline{X}$, and for every x,y,z in the domain X of $\underline{X}$

$$\underline{i}(x,y) = \big| g(x) - g(y) \big|$$

$$\underline{p}(x,y) = \max\big(0, g(x) - g(y)\big)$$

$$\underline{b}(x,y,z) = \tfrac{1}{2}\big(|g(x)-g(y)| + |g(y)-g(z)| - |g(x) - g(z)|\big) .$$

In this case g represents the (interpretations of) the three predicates under consideration in the way such predicates (especially preference, as in, e.g., Luce and Raiffa [5], p.29) are usually represented by utility functions. And we conclude by noting that the equations above are multi-valued analogues of the representations of, respectively, indifference graphs in Roberts [6], definite preferences ("semiorders") in Scott and Suppes [9] and tolerance betweenness in Roberts [7].

REFERENCES

[1] Giles, R., A non-classical logic for physics, Studia Logica, 33, 1974, 397-415.

[2] Katz, M., Inexact geometry, Notre-Dame Journal of Formal Logic, 21, 1980, 521-535.

[3] Katz, M., Two systems of multi-valued logic for science, Proceedings of the 11th International Symposium on Multiple-Valued Logic, IEEE Computer Society Press, New York, 1981, 175-182.

[4] Katz, M., The logic of approximation in quantum theory, Journal of Philosophical Logic, 11, 1982, 215-228.

[5] Luce, R.D. and Raiffa, H., Games and Decisions, John Wiley, New York, 1957.

[6] Roberts, F.S., Indifference graphs, in F. Harary (Ed.) Proof Techniques in Graph Theory, Academic Press, New York, 1969, 139-146.

[7] Roberts, F.S., Tolerance geometry, Notre-Dame Journal of Formal Logic, 14, 1973, 68-76.

[8] Scott, D.S., Completeness and axiomatizability in many-valued logic, in L. Henkin et al. (Eds.), Proceedings of the Tarski Symposium (Proceedings of Symposia in Pure Mathematics 25), American Mathematical Society, Providence, R.I. 1974, 188-197.

[9] Scott, D.S. and Suppes, P., Foundational aspects of theories of measurement, Journal of Symbolic Logic, 23, 1958, 113-128.

[10] Tarski, A., What is elementary geometry, in L. Henkin, P. Suppes and A. Tarski (Eds.), Symposium on the Axiomatic Method, North Holland, Amsterdam, 1959, 16-29.

[11] Tversky, A., Intransitivity of preferences, Psychological Review, 76, 1969, 31-48.

AN APPROACH TO FUZINESS IN THE SETTING OF ŁUKASIEWICZ LOGIC (*)

by Enric Trillas

Departament de Matemàtiques i Estadística (ETSAB)
Universitat Politècnica de Barcelona, Spain.

Abstract

The paper seeks to understand the meaning of
"fuzziness". Its aim is to show how in some cases
fuzziness comes from the indistinguishability bet-
ween a fuzzy set and its falsum, in a sense close
to Łukasiewicz ideas in Multiple-Valued Logic.
With such an aim the notion of "indistinguishabili-
ty relation" is considered and some partial results
are obtained.

1. Introduction

After the foundational paper (1) by A.DeLuca
and S.Termini a lot of papers on measures of fuzzi-
ness were produced (see references in (8)) special-
ly about the use of entropies and on the analysis
of its different types, but essentially no new ideas
appeared in the first eight years. In 1979 R.Yager
(7) introduced the idea that fuzziness comes from a
"lack of distinction" between a fuzzy set and its
complement $\overline{A} = 1-A$ in the setting of the classical
theory of fuzzy sets i.e., giving the union by the
max operator and the intersection by the min opera-
tor. In that approach fuzziness is measured by some
distances, which election is not clearly related
with such a lack of distinction, and although the
idea seems very interesting it pressuposes the exis-
tence of some crispness between A and $\overline{A}$, as if all
the region "neither not nor yes" were a broad bor-
der separating both the "yes" and "not" regions:for
example, to think what Yager says in the case of
fuzzy set of "round natural numbers", and specially
with "large round numbers", seems very difficult.
Yager's paper is a good first step towards a nice
idea that needs more work in the line of clarifying
the mathematical translation of "distinctness" and
its dependence on the distribution of the values of
the membership function (see (3)): it is not evident
that only one family of distances are predestinate
to measure fuzziness.

In the theory of fuzzy sets there is an essen-
tial difficulty with membership functions other than
the problem of determining it; that difficulty is
its non-uniqueness when membership function exists.
We will limite ourselves to fuzzy subsets $\underline{A}$ of X
that are representable by, at least, one characte-

ristic function $A:X \to [0,1]$, whatsoever the method
of obtaining it was. The study of fuzziness through
entropies in the sense of DeLuca-Termini supposes
that the measure of the fuzziness of $\underline{A}$ can be ob-
tained by knowing only one membership function A
(understanding by different membership functions
those that are pointwise different) that, always,
captures all the fuzziness of $\underline{A}$. Such pressuposition
is hardly realistic by the existence of $\underline{A}$ itself
and it is really difficult to assert that A and A'
do not represent the same $\underline{A}$ when the differences
$|A(x) - A'(x)|$ are small enough.

Perhaps, it is in the line of formulating ma-
thematically other kinds of "equality" (see (6))
where lies the possibility of founding the roots of
fuzziness and its meaning, by which entropies are
some sort of measures and, from another point of
view, where lies the real beginning of a new study
of vagueness and booleanity. As always, when measu-
res are needed the analysis of concepts became more
and more rigorous. This paper, only a preliminary
step towards this goal in the line of Yager, will
try to elucidate what the "distinguishability" of
two fuzzy sets at every point of X can be by intro-
ducing the thesis that, in some cases, fuzziness
can be considered as the "distinguishability" from
the so-called falsum that reflects the indistingui-
shability between $\underline{A}$ and the empty set $\emptyset$ and that,
with special restrictions, is coincidental with the
complement $\underline{\overline{A}}$.

2. Indistinguishability operators

Let X be a ground set of "forms" x,y,z,... and
L a set of "values" provided with a binary operation
* and a binary relation ≤. If it is convenient we
will suppose that (L,*) is a semigroup and (L,≤) a
partially ordered set or that (L,*,≤) is an ordered
semigroup. Let h ∈ L be a distinguished element and
E a map from XxX into L, such that
     1) $h \leq E(x,x)$
     2) $E(x,y) = E(y,x)$
     3) $E(x,y) * E(y,z) \leq E(x,z)$,
for any x, y and z in X. We call E an indistingui-
shability operator in X relative to L, * and ≤ at
the level h. We also talk about indistinguishabili-
ty relations.

Equivalences in ternary Logic by Łukasiewicz,
Bočvar, Kleene, Gödel, Reichenbach and Destouches-
Février are particular cases of indistinguishabili-
ty relations (see (5)).

Also, being H a complete Heyting Algebra, H-valued sets (11) on X are such kind of relations. In the case that X = L is a boolean Algebra is well known that the operator $E(x,y) = 1+x+y$ comes to be an indistinguishability operator, relative to both $(L, \leq)$ and $((L,+), (L,\leq))$, at the level h=1. If L is the boolean Algebra $P(X)$ of the crisp parts of X, it is $E(A,B) = X-A\Delta B$, i.e., the complementary set of its symmetrical difference.

If in L a strong negation function n exists, then the map $n\circ E : X\times X \to L$ is called a "distinguishability operator" in X. For example, in a boolean Algebra, with $n(x) = 1+x$, is $n\circ E(x,y) = x+y$. Those operators are, at it is well known, generalized distances (see (9) and (10)).

In what follows we will consider only the particular case in which between $*$ and $\leq$ there is a compatibility law and that h=1 is the neutral for $*$ and the maximum for the order $\leq$. In Fuzzy Sets literature the cases in which, being $L = [0,1]$ and $\leq$ the usual order of R, is either $* = \wedge = \min$ (Zadeh's SIMILARITIES (15)), or $* = \bullet = \text{prod}$ (Menger's PROBABILISTIC RELATIONS (12) and (13)) or $* = 0\vee(\text{sum}-1)$ (Ruspini's LIKENESS RELATIONS (14)) are well known. As it is $0\vee(a+b-1) \leq a.b \leq \min(a,b)$, it is obvious that with $0\vee(\text{sum}-1)$ we have the weaker case. After next paragraph we will deal only with $a \oplus b = \max(0,a+b-1)$ and $A \oplus B = \emptyset\cup(A+B-1)$ if $L = P(X) = [0,1]^X$.

The following two results are well known (see (5)).

Th. 2.1 E is a similarity relation if and only if 1-E is a pseudo-ultrametric bounded by 1.

Th. 2.2 E is a probabilistic relation if and only if
$$D(x,y) = \begin{cases} -\log E(x,y), & \text{if } E(x,y)\neq 0 \\ \infty, & \text{if } E(x,y)= 0 \end{cases}$$
is a pseudo-distance.

Next two assertions are easy to prove.

Th. 2.3 E is a likeness relation if and only if 1-E is a pseudo-distance bounded by 1.

Th. 2.4 E is a probabilistic relation bounded if and only if 1-E is a generalized metric relative to the ordered semigroup $([0,1], \leq; \text{sum} - \text{prod})$.

Those results, that do not need more comments, show the role of ordinary distances and of generalized metrics ((9),(10)).

3. On the case of Fuzzy Sets under the law

∅ ∪ (sum - 1).

If $A,B \in P(X)$ and E is an indistinguishability operator for $P(X)$ relative to some L,$E(A,B)$ shows the degree in which both A and B are indistinguishable. If we choose $L = P(X)$ and suitable operation and order, then $(A\approx B)(x) = E(A,B)(x)$ can be considered as the indistinguishability degree between A and B in the point $x \in X$ or, in Zadeh's terminology, the possibility of being A and B indistinguishable in $x \in X$. In what follows we will consider $\approx: P(X) \times P(X) \to P(X)$ verifying

1) $A \approx A = 1$
2) $A \approx B = B \approx A$
3) $(A \approx B) \oplus (B \approx C) \leq A \approx C$,

for any A, B and C in $P(X)$ and being $\leq$ the pointwise order, = the corresponding equality, and $\oplus$ the law $\emptyset \cup (\text{sum}-1)$ defined by
$$(A\oplus B)(x) = \max(0,A(x)+B(x)-1) = A(x)\oplus B(x).$$

Th. 3.1 A map $\approx:P(X) \times P(X) \to P(X)$ is an indistinguishability operator for $P(X)$, relative to $(P(X),\leq,\oplus;1)$ if and only if $d_x(A,B) = 1-(A\approx B)(x)$ are, for every $x \in X$, pseudo-distances bounded by 1.

Proof. Given $\approx$, d_x verifies $d_x(A,A) = 0$ and $d_x(A,B) = d_x(B,A)$, for any $x \in X$ and every A and B in $P(X)$. Moreover:
$$d_x(A,B)+d_x(B,C) = 1-((A\approx B)(x)+(B\approx C)(x)-1) \geq$$
$$1-\max(0,(A\approx B)(x)+(B\approx C)(x)-1) =$$
$$1-((A\approx B)\oplus(B\approx C))(x) \geq 1-(A\approx C)(x) = d_x(A,C),$$

then all the d_x are pseudo-distances, obviously bounded by 1. Reciprocally, if $\{d_x;x \in X\}$ are such kind of pseudo-distances, by definig $(A\approx B)(x) = 1-d_x(A,B)$ we have that $A\approx A = 1$ and $A\approx B = B\approx A$.
Moreover:
$$((A\approx B)\oplus(B\approx C))(x) = (A\approx B)(x)\oplus(B\approx C)(x) =$$
$$\max(0,1-d_x(A,B)-d_x(B,C)) =$$
$$1-\min(1,d_x(A,B)+d_x(B,C)) \leq 1-d_x(A,C) = (A\approx C)(x).$$

That theorem enable us to consider every indistinguishability operator in $P(X)$, relative to $(P(X),\leq , \oplus;1)$, as a family of co-distances $\{1-d_x;x\in X\}$ depending on the point.

As it is $[0,1] \subset P(X)$, via the identification between $t \subset [0,1]$ and the constant function $t(x)=t$, it is also $([0,1], \leq, \oplus;1)$ imbedded in $(P(X), \leq,\oplus;1)$ and we can consider the case in which the values of $\approx$ are in $[0,1]$ as a particular one formulated by the

Th. 3.2 A map $\approx:P(X) \times P(X) \to [0,1]$ is an indistinguishability operator for $P(X)$ relative to $([0,1], \leq, \oplus;1)$ if and only if $d(A,B) = 1-(A\approx B)$ is a pseudo-distance bounded by 1.

Proof. The same as in the theorem 3.1.

In that case indistinguishability operators are co-pseudo-distances.

If pseudo-distances are distances is
"$d_x(A,B) = 0, \forall x \in X$ iff $A=B$ iff $A(x)=B(x), \forall x \in X$".
Then:
"$A\approx B=1$ iff$(A\approx B)(x)=1,\forall x\in X$ iff $d_x(A,B)=0,\forall x\in X$ iff $A=B$",
and pointwise equality means the greatest indistinguishability. In any case, as it is $A\approx B = 1$ iff $d_x(A,B) = 0$, it is possible (if it is needed) to consider the classical equivalence "$A=B$ iff $A\approx B=1$".

Example 1. If $d_x(A,B) = |A(x)-B(x)|$, we have $(A\approx B)(x) = 1-|A(x)-B(x)|$ the so-called equivalence of Łukasiewicz, well known in MVL.

Example 2. If $d_x(A,B) = \dfrac{|A(x) - B(x)|}{1 + |A(x) - B(x)|}$, the Minkowski metrics, we have

$$(A \approx B)(x) = \frac{1}{1 + |A(x) - B(x)|}.$$

Example 3. Consider binary expansions

$$A(x) = \sum_{i=1}^{\infty} A(x)_i / 2^i, \quad A(x)_i \in \{0,1\}.$$

Then distances

$$d_x(A,B) = \sum_{i=1}^{\infty} \frac{|A(x)_i - B(x)_i|}{2^i}$$

are different from the euclidean and gives

$$(A \approx B)(x) = 1 - \sum_{i=1}^{\infty} \frac{|A(x)_i - B(x)_i|}{2^i}$$

as associated indistinguishability. We call it the "binary indistinguishability".

Example 4. If $d(A,B) = \sup_{x \in X} |A(x) - B(x)|$,

is $(A \approx B)(x) = \inf_{x \in X} (1 - |A(x) - B(x)|)$ the uniform indistinguishability.

Example 5. If X is finite, $X = \{x_1, \ldots, x_n\}$,

and $d(A,B) = \sqrt[p]{\sum_{i=1}^{n} (A(x_i) - B(x_i))^p}$, $p \geq 1$, we have

$$(A \approx B)(x) = 1 - \sqrt[p]{\sum_{i=1}^{n} (A(x_i) - B(x_i))^p},$$

the case $p=1$ corresponding to the Hamming distance and the $p=2$ to the Euclidean distance.

When, relatively to $(\underline{P}(X), \leq, \oplus; \underline{1})$ a functional expression $(A \approx B)(x) = E(A(x), B(x))$ is possible, the operator $E: [0,1] \times [0,1] \rightarrow [0,1]$ is an indistinguishability operator for $[0,1]$ relative to $([0,1], \leq, \oplus; 1)$. Conversely, if E is such an operator the preceding formula gives an operator for $\underline{P}(X)$ relative to $(\underline{P}(X), \leq, \oplus; \underline{1})$. Those are the cases with

$E(x,y) = 1 - |x-y|$ and $E(x,y) = \frac{|x-y|}{1+|x-y|}$. In any case $1 - E(A(x), B(x)) = d_x(A,B)$ are pseudo-distances in $\underline{P}(X)$.

As $1 - |A(x) - B(x)| = 1 - \max(A(x), B(x)) - \min(A(x), B(x))$ it is natural to wonder when it is possible to write $(A \approx B)(x) = 1 - (F'(A(x), B(x)) - F(A(x), B(x)))$, being F a t-norm and F' the corresponding dual t-conorm (10). From $F(t,t) \leq t \leq F'(t,t)$ and the fact that it is $(A \approx A)(x) = 1$ iff $F'(A(x), A(x)) = F(A(x), A(x))$, it follows $F'(A(x), A(x)) = F(A(x), A(x)) = A(x)$: In $[0,1]$ any number is idempotent by F, and $F = \min$. Then,

Th. 3.3 If F is a t-norm and F' its dual t-conorm,

$(A \approx B)(x) = 1 + F(A(x), B(x)) - F'(A(x), B(x))$
is an indistinguishability operator iff

$F = \min$ and $F' = \max$.

4. Fuzziness under the law $\emptyset \cup (\text{sum-1})$.

In Multiple-Valued Logic, Łukasiewicz defined the concept of *verum* as the equivalence between a proposition and the truth and the concept of *falsum* as the equivalence between a proposition and the false. We will consider only the *falsum* defined by:

$$(A \approx \emptyset)(x) = 1 - d_x(A, \emptyset)$$

that in the case of Łukasiewicz is $(A \approx \emptyset)(x) = 1 - A(x)$ or $A \approx \emptyset = \overline{A}$: the *falsum* coincides with the complement. This is not the situation with Minkowski distances, being $(A \approx \emptyset)(x) = 1 / 1 + A(x)$, fuzzy set that does not have the properties usually required for a "complement". With the binary indistinguishability it is

$$(A \approx \emptyset)(x) = 1 - \sum_{i=1}^{\infty} A(x)_i / 2^i = 1 - A(x),$$

that is $A \approx \emptyset = \overline{A}$.

To have $A \approx \emptyset = \overline{A}$ it is necessary that $d_x(A, \emptyset) = A(x)$. To such an end let us consider the following definition: A "translation" in $\underline{P}(X)$ is every map such that $t_B(A) = \underline{1} \cap (A+B)$, that is, such that it associates to every A the fuzzy set $(t_B(A))(x) = \min(1, A(x) + B(x))$. Let T be the set of all translations in $\underline{P}(X)$. As

$(t_C \circ t_B)(A) = t_C(t_B(A)) = t_C(\underline{1} \cap (A+B)) = \underline{1} \cap ((\underline{1} \cap (A+B)) + C)$,

we have,

 - If $A+B \geq \underline{1}$, it is $(t_C \circ t_B)(A) = 1$ and also $t_{B+C}(A) = 1$
 - If $A+B \leq \underline{1}$, it is $(t_C \circ t_B)(A) = t_{B+C}(A)$,

that is $t_C \circ t_B = t_{B+C}$ and T is a commutative semigroup. To have a group is it needed to enlarge $\underline{P}(X)$ till $\underline{P}'(X) = [-1,1]^X$. If it is necessary we will consider $\underline{P}'(X)$.

Minkowski metrics are not invariant under "translations" of T, being so under usual translations. In fact, it is enough to take A, B and C such that $A(x) + C(x) > 1$ and $B(x) + C(x) > 1$, for a $x \in X$, to have $d_x(t_C(A), t_C(B)) = 0$ but $d_x(A,B) \neq 0$.

Th. 4.1 The only distances invariant by "translations" of T and verifying $d_x(A, \emptyset) = A(x)$ are the Euclidean $d_x(A,B) = |A(x) - B(x)|$.

Proof. For a given $x \in X$ it is $|A(x) - B(x)| = A(x) - B(x)$ or $|A(x) - B(x)| = B(x) - A(x)$. In the first case, as $d_x(A-B, \emptyset) = A(x) - B(x)$ it is $d_x(A,B) = d_x(A-B+B, \emptyset+B) = d_x(t_B(A-B), t_B(\emptyset)) = d_x(A-B, \emptyset) = A(x) - B(x)$. In the second case, as $d_x(B-A, \emptyset) = B(x) - A(x)$, is it obtained $d_x(A,B) = d_x(B,A) = B(x) - A(x)$. Then the theorem holds.

It is a consequence of last theorem that "binary" distances are not invariant by translations. Note that those distances are greater than the Euclidean:

$$|A(x) - B(x)| = \left| \sum_{i=1}^{\infty} A(x)_i / 2^i - \sum_{i=1}^{\infty} B(x)_i / 2^i \right| \leq \sum_{i=1}^{\infty} \frac{|A(x)_i - B(x)_i|}{2^i}.$$

Such a situation is general.

Th. 4.2 Any distance d_x verifying $d_x(A, \emptyset) = A(x)$ is greater that the Euclidean.

Proof. It is

$$d_x(\emptyset,A)+d_x(A,B)\geqslant d_x(\emptyset,B)$$

and

$$d_x(\emptyset,B)+d_x(B,A)\geqslant d_x(\emptyset,A),$$

that is

$$d_x(A,B)\geqslant B(x)-A(x) \quad \text{and} \quad d_x(A,B)\geqslant A(x)-B(x),$$

implying

$$d_x(A,B)\geqslant \max(A(x)-B(x),B(x)-A(x)) = |A(x)-B(x)|.$$

Moreover, as $d_x(A,B)\leqslant d_x(A,\emptyset)+d_x(\emptyset,B)= A(x)+B(x),$

it is $d_x(A,B)\leqslant 1\wedge(A(x)+B(x)).$

Then, with the definition

$$D_x(A,B) = \begin{cases} 0, & \text{if } A=B \\ 1\wedge(A(x)+B(x)), & \text{if } A\neq B \end{cases}$$

we have a family of distances such that $D_x(A,\emptyset)=A(x)$ verifying

$$|A(x)-B(x)|\leqslant d_x(A,B)\leqslant D_x(A,B),$$

and giving the theorem

Th. 4.3   Every distance d_x verifying $d_x(A,\emptyset)=A(x)$ is less than the corresponding D_x.

In those conditions is immediate the general representation given by

Th. 4.4   If d_x is a family of distances verifying $d_x(A,\emptyset) = A(x)$, for any $x\in X$ and any couple A,B in $\underline{P}(X)$, there exists $\lambda^x_{AB}\in[0,1]$ such that

$$d_x(A,B)=\lambda^x_{AB}|A(x)-B(x)|+(1-\lambda^x_{AB})\cdot D_x(A,B).$$

Consequently, all the indistinguishability operators such that $A\approx\emptyset=\overline{A}$ are representable by

$$(A\approx B)(x)= 1-\lambda^x_{AB}|A(x)-B(x)|- (1-\lambda^x_{AB})\ D_x(A,B),$$

that is a very large family of indistinguishabilities. Those formulas are rarely useful due to the unknown function λ^x_{AB}.

Following the ideas of Yager (7) we will introduce the

Definition.   Given an indistinguishability operator $\approx$ in $\underline{P}(X)$, relative to $(\underline{P}(X),\leqslant,\oplus;1)$, the fuzzy set $\delta A = A\approx(A\approx\emptyset)$, is said to be the diffusum of $A\in\underline{P}(X)$, and it shows the indistinguishability degrees between A and its falsum. When the operator $\approx$ is such that $A\approx\emptyset=\overline{A}$ we will speak of "fuzziness" instead of "diffusum".

$$\text{Taking } d_x(A,B) = \sqrt[p]{\frac{\overset{\infty}{\underset{i=1}{\Sigma}} |A(x_i)-B(x_i)|^p}{n}}, \quad p\geqslant 1,$$

we have just the cases considered by Yager.

In what follows we will limite ourselves to the case of fuzziness. It is $\delta A= A\approx\overline{A}$ or

$$\delta A(x)=1-d_x(A,\overline{A})=1-\lambda^x_A|2A(x)-1|-(1-\lambda^x_A)\cdot 1$$

$= \lambda^x_A(1-2|A(x)-1/2|)$ if $A\neq 1/2$, and $\delta 1/2(x)=0,$

writing λ^x_A for $\lambda^x_{A\overline{A}}$. We call λ^x_A the specificity function of A.

The fuzziness corresponding to Euclidean distances $|A(x)-B(x)|$ is exactly $1-2|A(x)-1/2|$. We will write $\delta^E A(x)= 1-2|A(x)-1/2|$ and call that function the euclidean fuzziness of A.

Th. 4.5   The fuzziness of $A\in\underline{P}(X)$ is $\delta A(x)=\lambda^x_A\cdot\delta^E A(x)$, the product of the specificity function and the euclideanz fuzziness. It is

$$\delta A(x) = \begin{cases} 2\ \lambda^x_A\ \cdot A(x), & \text{if } A(x)\leqslant 1/2 \\ 2\ \lambda^x_A\ \cdot \overline{A}(x), & \text{if } 1/2 < A(x). \end{cases}$$

In any case it is $\delta A\leqslant\delta^E A$: the euclidean fuzziness is the greatest fuzziness under the law $\emptyset\cup(\text{sum-}1)$. Both fuzziness are coincidental if and only if the specificity function is equal to 1. On the other hand, it is $\delta A(x)= \lambda^x_A,\quad x\in X$, if and only if $\delta^E A(x)=1$, that happens only when $A=1/2$: the specificity function is the fuzziness only for the fuzzy set $1/2$.

Th. 4.6   The euclidean fuzziness δ^E verifies the properties on an entropy in the sense of DeLuca-Termini (but relatively to the structure of $\underline{P}(X)$).

Proof. 1) It is $\delta^E A=\emptyset$ iff $\delta^E A(x)= 0,\ \forall x\in X$, that happens only when $A(x)\in\{0,1\}$, i.e., when A is a crisp subset of X.

2) It is $\delta^E A=1$ iff $a=1/2$.

3) It is also a straightforward computation to prove that if $A\leqslant_s B$ (sharpened order (3)) then $\delta E_A\leqslant\delta E_B$ (pointwise order).

The formula $\delta A=\lambda^{(.)}_A\cdot\delta^E A$ shows that fuzziness can be a very sophisticated function, because of the specificity function that is an unknown function. The study of the properties of "entropy" for δA is more difficult that for $\delta^E A$: It is certainly, $\delta A=1$ iff $\lambda^{(.)}_A=\delta^E A=1$, that happens only if $A=1/2$, but to analyse its comportment relative to the sharpened order is necessary to know the comportment of the specificity function by the same order and, also for example, it can be $\delta A=\emptyset$ without being A crisp: it is enough for that that in some points x of X be $\lambda^x_A=0$ and $\delta^E A(x)\neq 0$ and in the remainder $\delta^E A(x)=0$.

The study of δA deserves a more acurate research and, for the time being, we will only consider the case in which distances d_x are in the convex hull

$$d_x(A,B)= \lambda\cdot|A(x)-B(x)|+(1-\lambda)\cdot D_x(A,B),\lambda\in[0,1].$$

In that case the specificity functions are constants and equal to λ, and $\delta A=\lambda\cdot\delta^E A$. Moreover,

Th. 4.7   When the specificity function is a constant, different from zero, the fuzziness δ verifies, relatively to the structure of $\underline{P}(X)$, the properties of an entropy of

225

DeLuca-Termini.

Proof. The proof runs as in theorem 4.6.

For every meaningful function $F:\underline{P}(X) \to [0,1]$ such that:

1) $F(A) = 0$ iff $A = \emptyset$
2) $F(A) = 1$ iff $A = \underline{1}$
3) $F(hA) = hF(A)$, if $h \in [0,1]$
4) If $A \leq B$ (pontwise), then $F(A) \leqslant F(B)$,

we have an "entropy" measuring the fuzziness of A by means of

$$F(\delta A) = \lambda \cdot F(\delta^E A),$$

as it is immediately shown as in the theorem 4.6. For example, being X finite, $X = \{x_1, x_2, \ldots, x_n\}$, with $F(A) = 1/n \cdot \sum_{i=1}^{n} A(x_i)$, it is

$$F(\delta A) = \lambda/n \cdot \sum_{i=1}^{n} (1-2|A(x_i)-1/2|) =$$

$$= \lambda(1-2/n \cdot \sum_{i=1}^{n} |A(x_i)-1/2|)$$

formula in which the parenthesis is well known "entropy" deduced from the Hamming distance (see(4)).

Of course, to measure the fuzziness, other mappings from $\underline{P}(X)$ into $[0,1]$ are possible. Note that, as $n.F(A)$ is exactly the "power" of the fuzzy set A, we may think, for example, on the "maximal degree" of A and use the function $g(A)=\sup_{x \in X} A(x)$ to measure the fuzziness by:

$$g(\delta A) = \lambda \cdot \sup_{x \in X} \delta^E A(x) = \lambda \cdot (1-2 \inf_{x \in X} |A(x)-1/2|),$$

although it does not satisgy all the properties of an entropy.

5. Concluding remark

The result stated in Th. 4.5 could be more interesting if it could be also extended to semigroups other than $([0,1], \oplus, \leqslant; 1)$. It is a conjecture that in a more general approach "fuzziness" can be obtained as a function of some sort of special fuzziness (generalizing the so-called euclidean fuzziness) and some parameter related to the given fuzzy set A (generalizing the so-called specificity of A).

References

(1) A. DeLuca and S. Termini (1972), "A definition of a nonprobabilistic entropy in the setting of fuzzy sets theory", INFORMATION AND CONTROL, 20 (4), 301-312.

(2) A. DeLuca and S. Termini (1979), Entropy and Energy Measures of a Fuzzy Set", in ADVANCES IN A FUZZY SET THEORY AND APPLICATIONS, M.M. Gupta, R.K.Ragade, R.R.Yager (editors),North Holland Pub., 321-338.

(3) E. Trillas and T. Riera (1978), "Entropies with Finite Fuzzy Sets", INFORMATION SCIENCES, 15, 159-168.

(4) E. Trillas and C. Sanchis (1979), "Sobre entropías de conjuntos borrosos deducidas de métricas", ESTADISTICA ESPAÑOLA, 82/83,17-25.

(5) E. Trillas (1982), "Assaig sobre les relacions d'indistingibilitat", ACTES DEL PRIMER CONGRES CATALA DE LOGICA MATEMATICA, 51-59, (Barcelona;Spain).

(6) E. Trillas (1982),"Sobre la igualdad de conjuntos borrosos", to be published in REVISTA DE LA REAL ACADEMIA DE CIENCIAS,LXXVI/4,895-899.

(7) R.R. Yager (1979),"On the measure of fuzziness and negation. Part I:Membership in the Unit Interval", INT.J.GENERAL SYSTEMS,5,221-229.

(8) E. Trillas and T. Riera (1982), "From measures of Fuzziness to Booleanity Control", in FUZZY INFORMATION AND DECISION PROCESSES,M. M. Gupta and E. Sanchez (editors), North Holland Pubs., 3-16.

(9) E. Trillas and C. Alsina (1979),"INTRODUCCION A LOS ESPACIOS METRICOS GENERALIZADOS", Serie Universitaria,49, Fund.March, Madrid, Spain.

(10) B. Schweizer and A. Sklar (1983), PROBABILISTIC METRIC SPACES, North Holland, New York, USA.

(11) M. P. Fourman and D.S. Scott (1979), "Sheaves and Logic" in APPLICATIONS OF SHEAVES, Lecture Notes in Math. 753, 302-401, Springer, Berlín.

(12) K. Menger (1951), "Probabilistic Theory of Relations", PROC.NAT.ACAD.SCI.USA, 37,178-180.

(13) S.V.Ovchinnikov(1982),"On Fuzzy Relational Systems",PROC.2nd.WORLD CON.ON MATH.AT THE SERVICE OF MAN,566-568 (Las Palmas , Spain).

(14) E.Ruspini(1982),"Recent Developments in Fuzzy Clustering",in FUZZY SET AND POSSIBILITY THEORY:RECENT DEVELOPMENTS,R.Yager (editor), Pergamon Press, 133-147.

(15) L.A. Zadeh (1971),"Similarity Relations and Fuzzy Orderings", INF.SCIENCES,3,177-200.

Invited Address

227

SYNTHESIS OF AXIOM SYSTEMS FOR THE THREE-VALUED
PREDICATE LOGIC BY MEANS OF THE SPECIAL FOUR-VALUED LOGIC

Motinori Goto and Shinji Kao and Tomoko Ninomiya

Meiji University, Tokyo, Japan

In the preceding papers [1] [2], M. Wajsberg's axiom system and their undefined operators ⊃ and 7 were treated as given logical equations and their unknown variables, where those solutions were indicated in the truth tables. By means of the similar method, Kleene's three-valued logic and Bochvar's one were also axiomatized. In this paper, above-mentioned axiom systems are extended to the complete predicate logic. To solve such simultaneous logical equations, the special four-valued logic is used.

1. introduction

In the recent mathematical logic, the operations of operators in axioms are not defined explicitly but done implicitly by many formulas derived from the axiom set. However, especially for many-valued axiom set, it is very laborious to derive the truth tables for those undefined operators.

In the preceding papers [1] [2], for example, undefined operators in M. Wajsberg's axiom set [2] were treated as unknown logical functions and axioms were treated as given logical equations. The general solution consists of truth tables (P

= 1 10) for the operators ⊃ and 7, which follow Wajsberg's axiom set (W₃) (6.5) (6.8), as shown in Table 1.1. Similarly Kleene's three-valued system (K₃) [3] and Bochvar's system (B₃) [4] were axiomatized, where their truth tables were derived as shown in Table 1.1 and Kleene's system consisted of three sets of truth tables.

In this paper, the coexistence of any two of "X = 1", "X = 2", "X = 3" is prohibited, as "Contradictory". Then each general solution for Wajsberg's, Kleene's or Bochvar's system converges to a single truth table (W'₃, K'₃, B'₃). The above-mentioned process of the solution of any given axiom set performed by means of the special four-valued logic [1] which is independent of the number of the truth-values of the given set.

Finally the axiom set for the predicate logic of the above-mentioned systems are also improved to obtain "Completeness".

REMARK 1.1 In this paper, the marks "!" and "!!" and "!!!" indicate "Assumption or Definition" and "Essential result" and "Axiom" respectively.

Table 1.1

	P		X = Y									X ⊃ Y									7X				
		X	1 1 1 2 2 2 3 3 3									1 1 1 2 2 2 3 3 3									1 2 3				
		Y	1 2 3 1 2 3 1 2 3									1 2 3 1 2 3 1 2 3									P				
REVISED												1 2 2 1 1 1 1 1 1									2 1 1		1		ORIGINAL
												1 2 2 1 1 1 1 1 1									3 1 1		2		
												1 2 3 1 1 1 1 1 1									2 1 1		3		
												1 2 3 1 1 1 1 1 1									3 1 1		4		
W₃'	1		1 3 3 3 1 3 3 3 1									1 2 3 1 1 2 1 1 1									3 2 1		5		W₃
												1 2 3 1 1 1 1 3 1									2 1 3		6		
												1 3 2 1 1 1 1 1 1									2 1 1		7		
												1 3 2 1 1 1 1 1 1									3 1 1		8		
												1 3 2 1 1 1 1 1 1									2 1 1		9		
												1 3 2 1 1 1 1 1 1									3 1 1		10		
			1 3 3 3 1 1 3 1 1									1 2 3 1 1 1 1 1 1									3 1 1		1		
			1 3 3 3 1 1 3 1 1									1 3 3 1 1 1 1 1 1									3 1 1		2		
K₃'	1		1 3 3 3 1 3 3 3 1									1 2 3 1 2 2 1 1 1									3 2 1		3		K₃
B₃'	1		1 3 3 3 1 3 3 3 1									1 2 3 2 2 2 1 2 1									3 2 1		1		B₃

2. General construction of axioms

Axioms are assumed to have the form "A ⊃ B" or "A = B". The logical expressions A and B consist of any variables X, Y, Z, ... of a finite number and their connectives ( ), ⊃, 7, and = of a finite number, where ( ) are parenthesis and 7 is an unary operator, while ⊃ and = are binary ones.

3. Truth-value

Every one of A, B, X, Y, Z, ... takes only one of the truth-values 1, 2, ..., N_0 simultaneously where "1" is designated as "True". In process of solution of logical equations, from Section 5 to 8, the following special expressions are used.

"X takes the truth-value x" is represented by $X^x = 1$. (3.1)
"X does not take x" is represented by $X^x = 0$. (3.2)

4. Connectives for classical two-valued logic

Negation A or ~ A.   Conjunction A·B.

Disjunction A ∨ B. Implication A → B. Equivalence A ↔ B.
Identity in many-valued logic A ≡ B, which means "Truth-values of A and B are equal for every truth-value of the constituents X, Y, Z, ...".

5. Representation of the undefined operators

Using the form (3.1) or (3.2), No-valued implication, equality and negation with truth value "ℓ" are expressed as the following two-valued expressions, where I^ℓ_{mn}, E^ℓ_{mn} and N^ℓ_m are indeterminate coefficients.

$$[A \supset B]^\ell \equiv IMP(A, B)^\ell \equiv \bigvee_{m=1}^{No} \bigvee_{n=1}^{No} I^\ell_{mn} \cdot A^m \cdot B^n. \quad (5.1)!$$

$$[A = B]^\ell \equiv EQU(A, B)^\ell \equiv \bigvee_{m=1}^{No} \bigvee_{n=1}^{No} E^\ell_{mn} \cdot A^m \cdot B^n. \quad (5.2)!$$

$$[7A]^\ell \equiv NEG(A)^\ell \equiv \bigvee_{m=1}^{No} N^\ell_m \cdot A^m. \quad (5.3)!$$

6. Basic tautologies for Wajsberg's three-valued logic

$$A^1 \cdot [A \supset B]^1 \to B^1. \quad (6.0)!$$

$$[X = X]^1. \quad (6.1)!!$$

$$[X = 77X]^1. \quad (6.2)!!!$$

$$[7(X = X) = 3]^1. \quad (6.3)!!!$$

$$[(X = Y) \supset ((Y = Z) \supset (Z = X))]^1. \quad (6.4)!!$$

$$[X \supset (Y \supset X)]^1. \quad (6.5)!!!$$

$$[(X \supset Y) \supset ((Y \supset Z) \supset (X \supset Z))]^1. \quad (6.6)!!$$

$$[(7Y \supset 7X) \supset (X \supset Y)]^1. \quad (6.7)!!!$$

$$[((X \supset 7X) \supset X) \supset X]^1. \quad (6.8)!!!$$

$$[(X = 7X) \supset (X = 2)]^1. \quad (6.9)!$$

$$[TX = 7TX]^1. \quad (6.10)!!!$$

where "!!!" indicates "Independent tautology" as proved in Table 10.1 and T is Slupecki's operator. [5]

7. Transformation of the tautologies in terms of I^ℓ_{mn}, E^ℓ_{mn}, N^ℓ_m

Substituting Eqs. (5.1) - (5.3) for the corresponding terms in Eqs. (6.0) - (6.8), we have Eqs. (7.0) – (7.8). For example, "1,J,K,1,m" in Eq. (6.7)

$$(\overset{\ell}{7}Y \overset{j}{\supset} \overset{m}{7}X) \overset{1}{\supset} (X \overset{k}{\supset} Y) \quad (7.7')$$

expresses the truth-values of the terms in Eq. (6.7) connected by operators "⊃, ⊃, ⊃, 7,7" respectively. Then, for any truth-values of x, y, z, we obtain by Eqs. (3.1) and (3.2).

$$(7.7') \equiv \bigwedge_{x,y=1}^{3} \bigvee_{j,k,\ell,m=1}^{3} I^1_{jk} \ I^j_{\ell m} \ N^\ell_y \ N^m_x \ I^k_{xy} \ X^x \ Y^y$$

$$\equiv \bigwedge_{x,y=1}^{3} \bigvee_{j,k,\ell,m=1}^{3} I^1_{jk} \ I^j_{\ell,m} \ N^\ell_y \ N^m_x \ I^k_{xy} \quad (7.7'')$$

without any other transformation, the sign · being omitted in Eqs. (7.2) (7.8). Eq. (6.0) is transformed to Eq. (7.0').

$$\overline{A^1} \vee IMP(A, B) \vee B^1 \equiv 1. \quad (7.0')$$

The 2nd term in the left-hand side should be 1 for A = 1, B ≠ 1. Then Eq. (7.0) is obtained.

$$\overline{I^1_{12}} \ \overline{I^1_{13}} \therefore I^1_{12} = I^1_{13} = 0, \quad (7.0)!!$$

$$\bigwedge_{x=1}^{3} E^1_{xx} = 1, \quad (7.1)!!$$

$$\bigwedge_{x=1}^{3} \bigvee_{j,k=1}^{3} E^1_{xj} \ N^j_k \ N^k_x = 1, \quad (7.2)!!$$

$$\bigwedge_{x=1}^{3} \bigvee_{j,k=1}^{3} E^1_{j3} \ N^j_k \ E^k_{xx} = 1, \quad (7.3)!!$$

$$\bigwedge_{x,y,z=1}^{3} \bigvee_{j,k,\ell,m=1}^{3} I^1_{jk} \ E^j_{xy} \ I^k_{\ell m} \ E^\ell_{yz} \ E^m_{zx} = 1, \quad (7.4)!!$$

$$\bigwedge_{x,y=1}^{3} \bigvee_{j=1}^{3} I^1_{xj} \ I^j_{yx} = 1, \quad (7.5)!!$$

$$\bigwedge_{x,y,z=1}^{3} \bigvee_{j,k,\ell,m=1}^{3} I^1_{jk} \ I^j_{xy} \ I^k_{\ell m} \ I^\ell_{yz} \ I^m_{xz} = 1, \quad (7.6)!!$$

$$\bigwedge_{x,y=1}^{3} \bigvee_{j,k,\ell,m=1}^{3} I^1_{jk} \ I^j_{\ell m} \ N^\ell_y \ N^m_x \ I^k_{xy} = 1, \quad (7.7)!!$$

$$\bigwedge_{x=1}^{3} \bigvee_{j,k,\ell=1}^{3} I^1_{jk} \ I^j_{kx} \ I^k_{x\ell} \ N^\ell_x = 1, \quad (7.8)!!$$

where in general

$$A^3_{mn} = \overline{A^1_{mn}} \cdot \overline{A^2_{mn}}, \ N^3_m = \overline{N^1_m} \cdot \overline{N^2_m}. \quad (7.00)!!$$

8. Solution of Eqs. (7.0) – (7.8) by a computer

Simultaneous logical Eqs. (7.0) – (7.8) are easily solved using FORTRAN by a computer. In FORTRAN, in place of "x, y, z, ...", we use corresponding capital letters.
For example,

$$I^\ell_{mn} = I(P, IBC), \qquad IBC = 9*L + 3*M + N. \quad (8.0.1)$$

$$E^\ell_{mn} = EQ(P, IBC), \ EBC = 9*L + 3*M + N \quad (8.0.2)$$

$$N^\ell_m = N(P, WA), \qquad NA = 9*L + 3*M. \quad (8.0.3)$$

$$P = 1, 2, ..., R.$$

8.1 Initial values of I^ℓ_{mn}

$$(7.0) \to I^1_{12} = I^1_{13} = 0. \quad (7.0)!$$

8.2 Initial values of E^ℓ_{mn}

The coexistence of "X=2" and "(X=1) or (X=3)" is assumed to be "Contradiction".

$$(6.4) \vdash \quad \left. \begin{array}{l} (X=1) \supset ((1=2) \supset (2=X)), \\ (X=2) \supset ((2=3) \supset (3=X)), \end{array} \right\} \quad (8.2.1)$$

$$\therefore 1 \neq 2, \ 2 \neq 3 \quad \therefore E^1_{12} = E^1_{23} = 0 \quad (8.2.2)!!$$

229

Similarly

$$E^1_{12} = E^1_{13} = E^1_{21} = E^1_{23} = E^1_{31} = E^1_{32},$$
$$= E^2_{12} = E^2_{13} = E^2_{21} = E^2_{23} = E^2_{31} = E^2_{12} = 0.$$

(8.2.3)!

The initial values of the remaining variables I^ℓ_{mn}, E^ℓ_{mn}, N^ℓ_m should be "2" as "Indeterminate". (8.2.4)!

8.3 Process of solution by means of the special 4-valued logic (8.3.2), (8.3.3)

In Table 8.1, XYZJK... B = 100 ... 0, means X=1 where 0's mean "The corresponding Y, Z, ..., B are nothing".

$$I(1 BC) = I^1_{bc}, I(2 BC) = I^2_{bc}, EQ(1BC) = E^1_{bc},$$
$$EQ(2BC) = E^2_{bc}, N(1A) = N^1_a, N(2A) = N^2_a.$$

(8.3.1.)

To combine many conditions, let us use the following special 4-valued logic (8.3.2) & (8.3.3).
Truth-value: $\phi, 0, 1, 2$.
"ϕ and "2" mean "Contradictory" and "Indeterminate".
"0" and "1" mean "False" and "True", respectively.

Conjunction
$$\phi \cdot \phi \equiv \phi \cdot 0 \equiv \phi \cdot 1 \equiv \phi \cdot 2 \equiv \phi,$$
$$0 \cdot 0 \equiv 0, 0 \cdot 1 \equiv \phi, 0 \cdot 2 \equiv 0,$$
$$1 \cdot 1 \equiv 1 \cdot 2 \equiv 1, 2 \cdot 2 \equiv 2.$$

(8.3.2.)!

Disjunction
$$\phi \vee \phi \equiv \phi, \phi \vee 0 \equiv 0, \phi \vee 1 \equiv 1, \phi \vee 2 \equiv 2,$$
$$0 \vee 0 \equiv 0, 0 \vee 1 \equiv 2, 0 \vee 2 \equiv 2,$$
$$1 \vee 1 \equiv 1, 1 \vee 2 \equiv 2, 2 \vee 2 \equiv 2.$$

(8.3.3)!

For the above-mentioned 4-valued operations, the following 2-bit expression may be used.

$$\phi = 00, 0 \equiv 01, 1 \equiv 10, 2 \equiv 11.$$ (8.3.4)!

We can easily derive (8.3.2) by the bit-wise conjunction of (8.3.4) and also (8.3.3) by the bit-wise disjunction of (8.3.4).
On the 1st row of Table (8.1). "20022 ... 222" is the 1st set (P1 = 1) of particular solutions which follow (7.0), (7.1), (7.00) for (X = 1).
On the 2nd row, there is the 1st set (P = 1) of the provisional solution produced by the conjunction of every initial value (8.2.3) and the corresponding values of the particular solution on the 1st raw.
On the 3rd raw, there is the 1st set (P1 = 1) of the particular solution which follows (7.0), (7.1), (7.00) for (X = 2).
On the 4th row, the 1st set (P = 1) of the provisional solution is derived by the conjunction of the preceding provisional solution (2nd row) and the preceding (3rd row) particular solution.
If "ϕ" is found in any new set, then that set should be cancelled and its ordinal number should be transferred to the next set.
As shown in Table 8.1, the truth values of I^ℓ_{mn}, E^ℓ_{mn}, N^ℓ_m are settled to "0" or "1" successively. Let us indicate such settled results as IO (IBC), EO (EBC) and NO (NA), which are derived by (8.3.5).

$$IO(IBC) = \bigvee_{P=1}^{R} I(P, IBC), EO(EBC) = \bigvee_{P=1}^{R} EQ(P, EBC),$$
$$NO(NA) = \bigvee_{P=1}^{R} N(P, NA),$$

(8.3.5)!!

where $\bigvee_{P=1}^{R}$ means the 4-valued disjunction (8.3.3) from (P = 1) to (P = R), R being the last ordinal number. Since the flow of computation proceeds unidirectionally from the first tautology to the last, any set of provisional solutions is the summary of all the preceding results.
The general solutions of I, EQ and N converge to the last single set of the provisional solution under the condition (8.2.3). This set is transformed to the truth-values of IMP(A, B), EQU(A, B) and NEG(A) according to Eqs. (8.3.6) and the truth table at the end of Table 8.1.

$$IMP(A, B) = 3 - 2*I(P, IAB) - I(P, IIAB),$$
$$EQU(A, B) = 3 - 2*EQ(P, IAB) - EQ(P, IIAB),$$
$$NEG(A) = 3 - 2*N(P, NA) - N(P, NNA)$$

(8.3.6)

where

$$IAB \equiv 9 + 3*A + B, \quad IIAB \equiv 9*2 + 3*A + B,$$
$$NA \equiv 9 + 3*A, \quad NNA \equiv 9*2 + 3*A.$$

9. Fittest number of truth-values and a canonical form

Table 9.1

X	7X	X1'	X2'	X3'
1	3	1	3	3
2	2	3	1	3
3	1	3	3	1

Table 9.1 shows the truth-values of the following three basic functions:

$$X1' \equiv (X = (X = X)), X2' \equiv (X = 7X), X3' \equiv (X = (X = 7X)),$$

(9.1)!

for W'_3, K'_3 and B'_3 in Table 1.1.

REMARK 9.1 $X1' \equiv X1$, $X2' \equiv X2$, $X3' \equiv X3$,

for K_3 & B_3 in [3], [4].

$$X1' = 7X1, X2' = 7X2, X3' = 7X3,$$ (9.2)!!

for W_3 in [2].

$$X \cup Y \equiv 7X \supset Y \text{ for } K'_3, K_3, B'_3, B_3,$$
$$X \cup Y \equiv (X \supset Y) \supset Y \text{ for } W'_3, W_3,$$
$$X \cap Y \equiv 7(7X \cup 7Y) \text{ for } W'_3, W_3, K'_3, K_3, B'_3, B_3,$$

(9.3)!

where the order of the operation is as follows:

$$7, \cap, \cup, \supset, \supset\subset, \equiv \atop =$$

As already reported in [6], [3], [4], we can divide the whole domain of X into three subdomains X1', X2', X3', which are displayed in Table 9.1.
If we use the operators $\cap$ and $\cup$ we can construct the canonical form F(X) which takes the truth-value Cn for (X = n).

Table 8.1

```
           I(1BC)     I(2BC)     EQ(1BC)   EQ(2BC) N(1A) N(2A)
A                                                  123 123
B          111222333 111222333  111222333 111222333
C          123123123 123123123  123123123 123123123
XYZJKLMHGFEDCB,        ,          ,          ,          ,      , P1, P
10000000000000, 200222222,222222222, 122222222,022222222, 222 222, 1
                200222222,222222222, 100020002,000020002, 222 222,    1
20000000000000, 200222222,222222222, 222212222,222202222, 222 222, 1
                200222222,222222222, 100010002,000000002, 222 222,    1
30000000000000, 200222222,222222222, 222222221,222222220, 222 222, 1
                200222222,222222222, 100010001,000000000, 222 222,    1
10011000000000, 200222222,222222222, 122222222,022222222, 122 022, 1
10012000000000, 200222222,222222222, 122222222,022222222, 012 102, 2
10013000000000, 200222222,222222222, 122222222,022222222, 021 020, 3
                200222222,222222222, 100010001,000000000, 122 022,    1
                200222222,222222222, 100010001,000000000, 012 102,    2
                200222222,222222222, 100010001,000000000, 021 020,    3
20021000000000, 200222222,222222222, 222212222,222202222, 012 102, 1
20022000000000, 200222222,222222222, 222212222,222202222, 202 212, 2
20023000000000, 200222222,2?2222222, 222212222,222202222, 200 201, 3
                200222222,222222222, 100010001,000000000, 102 012,    1
                200222222,222222222, 100010001,000000000, 100 001,    2
                200222222,222222222, 100010001,000000000, 012 102,    3
                200222222,222222222, 100010001,000000000, 001 010,    4
30031000000000, 200222222,222222222, 222222221,222222220, 021 020, 1
30032000000000, 200222222,222222222, 222222221,222222220, 200 201, 2
30033000000000, 200222222,222222222, 222222221,222222220, 220 220, 3
                200222222,222222222, 100010001,000000000, 100 010,    1
                200222222,222222222, 100010001,000000000, 100 001,    2
                200222222,222222222, 100010001,000000000, 010 100,    3
                200222222,222222222, 100010001,000000000, 001 010,    4


                                omitted


23022120000000, 200210222,212201222, 222222222,222222222, 201 210, 1
                100110111,011001000, 100010001,000000000, 001 010,    1
                100110111,010001000, 100010001,000000000, 001 010,    2
31011310000000, 100222122,022222022, 222222222,222222222, 021 020, 1
                100110111,011001000, 100010001,000000000, 001 010,    1
                100110111,010001000, 100010001,000000000, 001 010,    2
32011210000000, 100122212,022022202, 222222222,222222222, 201 210, 1
                100110111,011001000, 100010001,000000000, 001 010,    1
                100110111,010001000, 100010001,000000000, 001 010,    2
33011110000000, 100222221,022222220, 222222222,222222222, 221 220, 1
                100110111,011001000, 100010001,000000000, 001 010,    1
                100110111,010001000, 100010001,000000000, 001 010,    2
10012300000000, 100122222,021022222, 222222222,222222222, 022 022, 1
10013300000000, 100222122,020222022, 222222222,222222222, 022 022, 2
                100110111,011001000, 100010001,000000000, 001 010,    1
                100110111,010001000, 100010001,000000000, 001 010,    2
20021200000000, 200212222,212202222, 222222222,222222222, 202 212, 1
                100110111,011001000, 100010001,000000000, 001 010,    1
                100110111,010001000, 100010001,000000000, 001 010,    2
30031100000000, 200222121,220222020, 222222222,222222222, 221 220, 1
                100110111,010001000, 100010001,000000000, 001 010,    1


            IMP(A,B)  EQU(A,B) NEG(A)

        A 111222333 111222333 123   P
        B 123123123 123123123
TRUTH VALUE
          123112111 133313331 321   1
```

$$F(X) \equiv (X1' \cap C1) \cup (X2' \cap C2) \cup (X3' \cap C3). \qquad (9.4)!$$

In this case:

$$F(n) = Cn. \qquad (9.5)!!$$

When Axiom (6.10) or constant 2 is not used, "C2=2" appears only when X=2, then we may use X in place of C2 in such case.

Table 9.2

	X∪Y									X∩Y								
X	1	1	1	2	2	2	3	3	3	1	1	1	2	2	2	3	3	3
Y	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
W_3'	1	1	1	1	2	2	1	2	3	1	2	3	2	2	3	3	3	3
K_3'	1	1	1	1	2	2	1	2	3	1	2	3	2	2	3	3	3	3
B_3'	1	2	1	2	2	2	1	2	3	1	2	3	2	2	2	3	2	3

If there is Cn = 2 when Axiom (6.10) is used, we may use TX in place of that Cn.

For F (X, Y), we may use F_n (Y) in place of Cn.

10. Selection of the independent axioms for W_3'

Let us select the independent tautologies among Eqs. (6.1) – (6.8). Table 10.1 shows the result of the selection, where "MS" indicates the ordinal number of the tautology and "R" does the cardinal number of the sets of provisional solutions at that tautology and "⊙" does "Independent tautology" and "△" does "This is not independent. "○" and "△" indicate "Already" decided as "independent" and "Already decided as dependent" respectively. Then, Axioms (6.2) – (6.8) give the unique solution for W_3'.

Table 10.1

MS	1	2	3	4	5	6	7	⊙8
R	1	4	1	2	16	12	2	1
MS	1	2	3	4	5	6	○8	⊙7
R	1	4	1	2	16	12	9	1
MS	1	2	3	4	5	○8	○7	△6
R	1	4	1	2	16	9	1	1
MS	1	2	3	4	○8	○7	⊙5	△6
R	1	4	1	2	6	4	1	1
MS	1	2	3	○5	○8	○7	△4	△6
R	1	4	1	20	11	1	1	1
MS	1	2	○7	○5	○8	⊙3	△4	△6
R	1	4	180	4	2	1	1	1
MS	1	○3	○7	○5	○8	⊙2	△4	△6
R	1	1	235	10	4	1	1	1
MS	○2	○3	○7	○5	○8	△1	△4	△6
R	4	1	40	2	1	1	1	1

11. Tautologies for K_3' and B_3'

As already reported [3], [4], independent axiom systems (!!!) for K_3' and B_3' are as follows.

$$A_!^!(A \supset B)^! \to B_!^!$$

[K_3']

$[X=X]!$	!!
$[(X=Y)=7(X=Y))\supset Z]!$	!!!
$[7(X=Y)\supset((X=Y)\supset Z)]!$	!!
$[X=Y)\supset((Y=Z)\supset(Z=X))]!$	!!!
$[7X\supset Y=7Y\supset X]!$	!!
$[7(X=7X)\supset((X=Y)\supset(X\supset Y))]!$	!!!
$[(X=7X)\supset((7X\supset Y)\supset 7(X\supset 7Y)=X)]!$	!!
$[(X=7X)\supset((Y=7Y)\supset(X=Y))]!$	!!
$[X=(X\supset 7Y)\supset X]!$	!!!
$[(7X=7Y)\supset(X=Y)]!$	!!!
$[X=77X].$	!!

[B_3']

$[X=X]!$	!!
$[X=77X]!$	!!
$[7((X=Y)=7(X=Y))]!$	!!
$[(X=Y)\supset((Y=Z)\supset(Z=X))]!$	!!!
$[7X\supset Y=7Y\supset X]!$	!!
$[(7X=7Y)=(X=Y)]!$	!!
$[(X=7X)\supset(X\supset Y=Z)]!$	!!!
$[7(X=7X)\supset(Y=(Y\supset 7X)\supset Y)]!$	!!!
$[X\supset 7Y=Y\supset 7X]!$	!!
$[X\supset Y=7Y\supset 7X]!$	!!!

12. Predicate logic
12.1 Classical two-valued predicate logic

Usually the following two axioms are used for classical two-valued predicate logic

$$\forall x P(x) \to P, \quad (12.1)!!! \qquad P \to \exists x P(x). \qquad (12.2)!!!$$

For simplicity, let us use the following expressions.

$$P \equiv P(Y_n) \equiv P_n, \quad \forall \equiv AxP(x), \quad \exists \equiv ExP(x). \qquad (12.3)!$$

Then (12.1), (12.3) ⊢

$$\bigwedge_n (\forall \to P_n) \equiv \bigwedge_n (\sim \forall \vee P_n) \equiv \sim \forall \vee \bigwedge_n P_n \cdot \forall$$

$$\equiv (\forall \longleftrightarrow 0) \cdot (\bigwedge_n P_n \longleftrightarrow 0) \vee (\forall \longleftrightarrow 0) \overset{\blacktriangle}{\cdot} (\bigwedge_n P_n \longleftrightarrow 1) \vee (\forall \longleftrightarrow 1) \cdot (\bigwedge_n P_n \longleftrightarrow 1).$$

$$(12.1.1)!!$$

(12.2), (12.3) ⊢ $\bigwedge_n (P_n \to \exists) \equiv \bigwedge_n (\sim P_n \vee \exists) \equiv \bigwedge_n \sim P_n \cdot \sim \exists \vee \exists$

$$\equiv (\exists \longleftrightarrow 1) \cdot (\bigvee_n P_n \longleftrightarrow 1) \vee (\exists \longleftrightarrow 1) \overset{\blacktriangle}{\cdot} (\bigvee_n P_n \longleftrightarrow 0) \vee (\exists \longleftrightarrow 0) \cdot (\bigvee_n P_n \longleftrightarrow 0).$$

$$(12.2.1)!!$$

In these general solutions (12.1.1) and (12.2.1), the special terms with Mark ▲ contradict the usual meaning.

Then we add the following axioms which exclude such unfavorable solutions ▲.

$$[P(y) \equiv 1] \longleftrightarrow \bigwedge_x (P(x) \longleftrightarrow (x \longleftrightarrow x)). \tag{12.4}!!!$$

$$(P \equiv 1) \to \forall x P(x). \quad (12.5)!!! \quad (\sim P \equiv 1) \to \sim \exists x P(x). \quad (12.6)!!!$$

12.2 Predicate logic for W_3', K_3', B_3'

Using (1.2.3), any representative "P_1 or P_2 or P_3" of P takes value "1 or 2 or 3".

For W_3' in Table 1.1, (12.1) has the following general solution, where the Marks ▲ indicate the unfavorable solution as in (12.1.1).

$$[\forall x P(x) \supset P]' \equiv [\forall \supset P]' \equiv [\bigwedge_n (\forall \supset P_n) = 1]' \equiv [\forall \supset \bigwedge_n P_n = 1]'$$

$$\equiv [\forall = 3] \cdot [\bigwedge_n P_n = 3] \vee [\forall = 3] \overset{\blacktriangle}{\cdot} [\bigwedge_n P_n = 2] \vee [\forall = 3] \overset{\blacktriangle}{\cdot} [\bigwedge_n P_n = 1]$$

$$\vee [\forall = 2] \cdot [\bigwedge_n P_n = 2] \vee [\forall = 2] \overset{\blacktriangle}{\cdot} [\bigwedge_n P_n = 1] \vee [\forall = 1] \cdot [\bigwedge_n P_n = 1].$$

$$(12.7)!!$$

Similarly, Eq. (12.2) has the general solution (12.8).

$$[P \supset \exists x P(x)]' \equiv [P \supset \exists]' \equiv [\bigwedge_n (P_n \supset \exists) = 1]' \equiv [\bigvee_n P_n \supset \exists = 1]'$$

$$\equiv [\exists = 1] \cdot [\bigvee_n P_n = 1] \vee [\exists = 1] \overset{\blacktriangle}{\cdot} [\bigvee_n P_n = 2] \vee [\exists = 1] \overset{\blacktriangle}{\cdot} [\bigvee_n P_n = 3]$$

$$\vee [\exists = 2] \cdot [\bigvee_n P_n = 2] \vee [\exists = 2] \overset{\blacktriangle}{\cdot} [\bigvee_n P_n = 3] \vee [\exists = 3] \cdot [\bigvee_n P_n = 3].$$

$$(12.8)!!$$

Then, instead of (12.1) and (12.2) the following new axioms should be used as shown in Table 12.1.

13. Conclusion

In this paper, M. Wajsberg's axiom system is improved to the complete system by means of the special 4-valued logic. Similarly Kleene's 3-valued logic and Bochvar's one are also axiomatized to the complete systems. Finally those three systems are extended to the predicate logic systems. Such method is applicable to other multiple-valued logic systems.

Table 12.1

	Axioms	Derived general solutions
	$(P(Y) \equiv 1) = \bigwedge_x [P(x) = (x = x)]'$	
	$[(P \equiv 1) \supset \forall]'.$	$(\bigwedge_n P_n = 1) \cdot (\forall = 1).$
	$[(P = (P = 7P)) \supset (\forall = P)]'.$	$(\bigwedge_n P_n = 3) \cdot (\forall = 3).$
	$[(7 (P = (P = 7P)) \equiv 1)$	
W_3'	$\supset ((P = 7P) \supset (\forall = P))]'.$	$(\bigwedge_n P_n = 2) \cdot (\forall = 2).$
&		
K_3'	$[(7P \equiv 1) \supset 7\exists]'.$	$(\bigvee_n P_n = 3) \cdot (\exists = 3).$
	$[(P = (P = P)) \supset (\exists = P)]'.$	$(\bigvee_n P_n = 1) \cdot (\exists = 1).$
	$[(7 (P = (P = P)) \equiv 1)$	
	$\supset ((P = 7P) \supset (\exists = P))]'.$	$(\bigvee_n P_n = 2) \cdot (\exists = 2).$
	$[(P \equiv 1) \supset \forall]'.$	$(\bigwedge_n P_n = 1) \cdot (\forall = 1).$
	$[(7 (P = 7P) \equiv 1) \supset (\forall \supset P)]'.$	$[P_n \neq 2] \cdot [(\bigwedge_n P_n = 1) \cdot (\forall = 1)$
		$\vee (\bigvee_n P_n = 3) \cdot (\forall = 3)].$
	$[(P = 7P) \supset (\forall = P)]'.$	$(\bigwedge_n P_n = 2) \cdot (\forall = 2).$
B_3'		
	$[(7P \equiv 1) \supset 7\exists]'.$	$(\bigvee_n P_n = 3) \cdot (\exists = 3).$
	$[(7 (P = 7P) \equiv 1) \supset (P \supset \exists)]'.$	$[P_n \neq 2] \cdot [(\bigvee_n P_n = 1) \cdot (\exists = 1)$
		$\vee (\bigvee_n P_n = 3) \cdot (\exists = 3)].$
	$[(P = 7P) \supset (\exists = P)]'.$	$(\bigvee_n P_n = 2) \cdot (\exists = 2).$

233

References

[1] Motinori Goto: Various types of General Solutions of Many-Valued Logical Equations with Many Variables. Bulletin of the Electrotechnical Laboratory (ETL), Vol. 20, No. 9 (Tokyo, Japan, Sept. 1956).

[2] Motinori Goto, Shinji Kao, T. Ninomiya: Determination of Many-Valued Truth Tables for Undefined Operators in Axioms by a Computer and Their Applications. Proceedings of the Seventh International Symposium on Multiple-Valued Logic. May, 1977. pp. 20   28.

[3] M. Goto, S. Kao, T. Ninomiya: Axiomatization of Kleene's Three-Valued Logic by a Computer. Proceedings of the Ninth International Symposium on Multiple-Valued Logic. May, 1979. pp. 241   247.

[4] M. Goto, S. Kao, T. Ninomiya: Axiomatization of Bochvar's Three-Valued Logic by a Computer. Proceedings of the 11th International Symposium on Multiple-Valued Logic. May 1981. pp. 146 - 151.

[5] Nicholas Rescher: Many-Valued Logic, Mc Graw-Hill (1969)

[6] M. Goto, S. Kao, T. Ninomiya: Determination of the Fittest Number of Truth-Values and Canonical Forms of Logical Functions for a Many-Valued Axiom Set by a Computer. Proc. of the 8th International Symposium on Multiple-Valued Logic. May, 1978. pp. 195   201.

Table of Errata in [4]

page	for	read
150 left 10 from the bottom	indicates the ordinal number of	indicates the ordinal number of the tautology and R does the cardinal number of

234

Session 6A
System Design and Applications

IMAGE PROCESSING ALGORITHMS FOR A MULTIPLE-VALUED ARRAY PROCESSOR

Michitaka KAMEYAMA, Kenichi SUZUKI and Tatsuo HIGUCHI

Department of Electronic Engineering, Faculty of Engineering
Tohoku University, Aoba, Aramaki, Sendai 980, Japan

ABSTRACT

A new digital image processor called multiple-valued array processor (MVAP) is effectively employed for systematic image processing without encoding and decoding because each pixel can be directly expressed by a single multiple-valued digit in the images with gray levels of several colors. In this paper, some properties of image processing in the MVAP are presented. Especially, the near-neighbor instructions can be attributed to template matching as the state transition function. In multiple-valued logic, there exist many templates because of its logical richness. The simplification of the state transition function is very useful for the effective execution of the image processing. The systematic design method of the image processing algorithm and its simplification using a minimization technique of multiple-valued logic functions are discussed.

I.   INTRODUCTION

In recent years, a great deal of time and effort has been expended in the field of image processing. Image processing machines are used to analyze satellite pictures, count blood cells, analyze histological sections, and process other forms of image data. Many approaches have been taken from both the hardware and the software in image processor design [1-5].

Until now, image processor designs have been based on binary logic circuits. The use of multiple-valued logic in image processing was first introduced by Rine [6, 7]. If the maximum number of gray levels or distinct colors is set at r, then each pixel can be represented by a single r-valued digit. In this case, digital encodings and decodings of the images are eliminated. Moreover, the complexity of the image processing algorithms is reduced due to reduction in the number of pixel iterations. From this point of view, we have designed the multiple-valued array processor (MVAP) which is an extension of the BASE [5, 8]. The MVAP can directly accept the pixels of the input image, process them and produce the output without converting back and forth between the actual image and the binary data.

In the MVAP, multiple-valued logical operations and near-neighbor instructions can be performed without encoding and decoding. The near-neighbor instructions can be generalized by template matching. The r-valued logic system has much more logical functions than the binary logic system. This logical richness and powerfulness in the MVAP makes image processing more and more efficient and flexible than in the binary array processor (BAP). On the other hand, the number of state transitions corresponding to the templates becomes large in the multiple-valued logic system because of its logical variety. Therefore, simplification of the image processing algorithms is required for effective execution, implying the minimum number of execution steps. For this purpose, a systematic synthesis method of the image processing algorithms is discussed. Finally, it is demostrated that the method is highly useful for the simplification of image processing algorithms in the MVAP.

II.   OVERVIEW OF THE MVAP

The basic structure of the subprocessor in the MVAP is shown in Fig. 1. The subprocessors are arranged in each pixel (i, j), and their operations are controlled by microprograms. The value of one of the inputs is transmitted to the eight near-neighbor subprocessors (N_1, N_2, ..., N_8) defined by Fig. 2. The control vector $g = (g_1, g_2, ..., g_8)$ determines which directions are permitted to infulence the variable P expressed by

$$P = \bigvee_{i=1}^{8} (g_i \wedge N_i) \qquad (1)$$

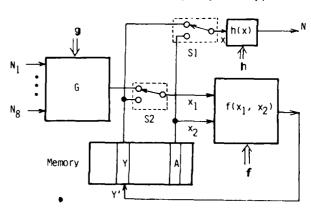where $\vee$ and $\wedge$ are OR and AND, respectively, and

Fig. 1   Subprocessor structure of MVAP

they are defined by

$$\bigvee : \quad OR(x_1, x_2) = \max(x_1, x_2)$$
$$\bigwedge : \quad AND(x_1, x_2) = \min(x_1, x_2) \quad (2)$$

$$\text{for } x_1, x_2 \in L = \{0, 1, \ldots, r-1\}.$$

The variable g_i is defined by

$$g_i = \begin{cases} r-1 & \text{if } N_i \text{ is selected} \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

As a basic building block to construct any combinational circuit, the multiple-valued T-gate is useful because of its universality [9, 10]. The multiple-valued T-gate which is a multiplexor function is defined by

$$T(p_0, p_1, \ldots, p_{r-1} ; x) = p_i \text{ if } x = i \quad (4)$$

where p_i $(i = 0, \ldots, r-1) \in L$ and $x \in L$. Any 2-variable functions can be expressed in the following canonical form:

$$f(x_1, x_2) = T(f(x_1, 0), \ldots, f(x_1, r-1) ; x_2)$$
$$= T(T(f(0, 0), \ldots, f(r-1, 0) ; x_1),$$
$$T(f(0, 1), \ldots, f(r-1, 1) ; x_1),$$
$$\ldots, T(f(0, r-1), \ldots, f(r-1, r-1)$$
$$; x_1) ; x_2). \quad (5)$$

Various near-neighbor instructions can be performed by the multiple-valued logic circuit $f(x_1, x_2)$ as well as two-term operations.

The different types of instructions are selected by the switches S1 and S2. The switch S2 selects either a multiple-valued or a near-neighbor instruction, and the switch S1 selects either a simple near-neighbor or a recursive near-neighbor instruction. The control vector $\mathbf{h}$ enables the propagation signal to be transformed according to the mapping. That is, h(x) can realize any single-term operation.

III.   IMAGE PROCESSING ALGORITHM

(1) Multiple-valued logical operation

In the r-valued logic system, the number of single-term operators and two-term operators are given by r^r and r^{r^2}, respectively. This logical power and richness enables the subprocessor to execute various types of image processing. The structure of the subprocessor to realize multiple-valued logical operation is shown in Fig. 3. As an example, let us consider the difference of two input images. The logical

difference called "MEXOR" is given by [6]

$$f(x_1, x_2) \equiv |x_1 - x_2| = T(x_2, T(1, 0, 1, 2 ; x_2),$$
$$T(2, 1, 0, 1 ; x_2), T(3, 2, 1, 0 ; x_2) ; x_1). \quad (6)$$

Each element of the set $L = \{0, 1, 2, 3\}$ corresponds to a symbol of Table I. For the input images of A and B, applying Eq. (6) the resultant image can be obtained as shown in Fig. 4.

(2) Simple near-neighbor instruction

Simple near-neighbor instructions use the interconnections between the subprocessors, and their structure for each subprocessor is shown in Fig. 5. The near-neighbor function is specified by

$$P \leftarrow NN(<list>) \text{ of } h(X) \text{ Edge}<value> \quad (7)$$
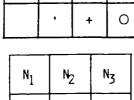
where X is the name of the multiple-valued array being operated on, and the notation $<list> \subset \{1, 2, 3, 4, 5, 6, 7, 8\}$ is a vector indicating which of the near neighbors are to be involved. The notation $<value> \in L$ indicates the value of near-neighbor inputs to the subprocessor at the edge of the array. If the edge is not specified, all edge elements are set to 0. Once the variable P has been evaluated, it is then combined with the other variable by two-input logic function to form the resultant multiple-valued array R. The complete notation for this instruction is

$$R \leftarrow f(A, P) \text{ where } P \leftarrow NN(<list>) \text{ of } h(A)$$
$$\text{Edge}<value> \quad (8)$$

Table I   Symbol of each pixel in the 4-valued logic system

0	1	2	3
	.	+	O
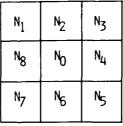
Fig. 2   Near-neighbor labeling

N_1	N_2	N_3
N_8	N_0	N_4
N_7	N_6	N_5
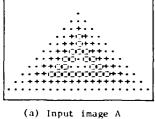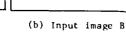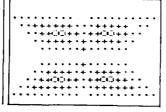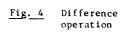
Fig. 3   Structure of the multiple-valued operations (2-term operations)

(a) Input image A

(b) Input image B

(c) Difference

Fig. 4   Difference operation

237

where f is any r-valued function of its arguments.

The simple near-neighbor instructions are generalized by the template matching. The transition function should be defined by a list of only those neighborhoods that actually will produce a change of the state in the center module together with the new state where the transition will be made [4]. The following notation will be used for the transition δ:

$$\delta: \begin{vmatrix} v_1^1 & v_2^1 & v_3^1 \\ v_8^1 & v_0^1 & v_4^1 \\ v_7^1 & v_6^1 & v_5^1 \end{vmatrix} \rightarrow v^1, \ldots, \begin{vmatrix} v_1^k & v_2^k & v_3^k \\ v_8^k & v_0^k & v_4^k \\ v_7^k & v_6^k & v_5^k \end{vmatrix} \rightarrow v^k \quad (9)$$

It is interpreted in the following way. If for any μ, $1 \leq \mu \leq k$, $A(i, j) = v_0^\mu$, $A(i-1, j-1) = v_1^\mu$, ..., and $A(i-1, j) = v_8^\mu$, then the result $R(i, j) = v^\mu$ or else $R(i, j) = A(i, j)$. As one of the templates in Eq. (9), consider the template given by

$$\begin{vmatrix} p_1 & p_2 & p_3 \\ p_8 & p_0 & p_4 \\ p_7 & p_6 & p_5 \end{vmatrix} \rightarrow p \quad (10)$$

If the near-neighbor input variables are defined as shown in Fig. 6, the product term corresponding to Eq. (10) becomes

$$p = x_0^{p_0} \cdot x_1^{p_1} \cdot x_2^{p_2} \cdot x_3^{p_3} \cdot x_4^{p_4} \cdot x_5^{p_5} \cdot x_6^{p_6} \cdot x_7^{p_7} \cdot x_8^{p_8} \quad (11)$$

where $\cdot$ denotes AND, and where $x_i^{p_i}$ is a literal defined by

$$x_i^{p_i} = \begin{cases} r-1 & \text{if } x_i = p_i \\ 0 & \text{otherwise.} \end{cases} \quad (12)$$

The complement of Eq. (11) is equivalent to

$$R = \overline{p} = \overline{x_0^{p_0}} + \overline{x_1^{p_1}} + \overline{x_2^{p_2}} + \overline{x_3^{p_3}} + \overline{x_4^{p_4}} + \overline{x_5^{p_5}} + \overline{x_6^{p_6}} + \overline{x_7^{p_7}} + \overline{x_8^{p_8}} \quad (13)$$

where $\overline{x}$ means the complement of x such that $\overline{x} = r-1-x$. The sum terms in multiple-valued logic are used for the execution of the template matching at each step because the near-neighbor function P is the logical sum of the near neighborhoods. Let the set of all the near neighborhoods such that $p_i = c$ be $\{I_c\}$ for $c \in L$. With respect to the near neighborhoods $\{I_c\}$, the template matching for the input A can be performed as

$$R(\text{Initial}) = 0 \quad (14)$$

$$R \leftarrow OR(P, R) \text{ where } P \leftarrow NN(<I_c>) \text{ of } \overline{A^c}. \quad (15)$$

If the result remains 0, after the iteration of the operation in Eq. (15) from $c = 0$ to $c = r-1$, then all the near neighborhoods are matched with the template. Therefore, the result of the template matching is obtained by

$$R \leftarrow f(R, A) \quad (16)$$
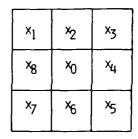
where $f(0, p_0) = p$, otherwise $f(R, A) = A$.

(3) Recursive near-neighbor instruction
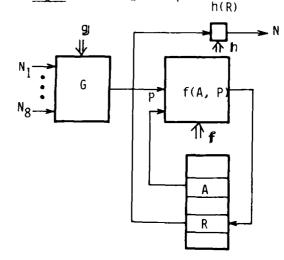
The recursive near-neighbor instruction can be implemented by the subprocessor structure shown in Fig. 7. The general notation for the recursive near-neighbor instruction is given by

R(Initial); given

Repeat R $\leftarrow$ f(A, P)
   where P $\leftarrow$
   NN(<list>) of
   h(R) Edge<value>.
   (17)

This instruction is also represented by the state transition of R using templates as follows:

x_1	x_2	x_3
x_8	x_0	x_4
x_7	x_6	x_5

<u>Fig. 6</u>   Near-neighbor inputs

<u>Fig. 5</u>   Structure of the simple near-neighbor instrucitons

<u>Fig. 7</u>   Structure of the recursive near-neighbor instructions

R(Present state)   A(Input)   R(Next state)

$$\begin{bmatrix} r_1 & r_2 & r_3 \\ r_8 & r_0 & r_4 \\ r_7 & r_6 & r_5 \end{bmatrix} \qquad a_0 \longrightarrow r_0' \qquad (18)$$

where a_0 and r_0' are the center pixels in the arrays A and R, respectively. The recursive instruction is effectively repeated until R reaches a constant value.

The sufficient conditions for the convergence of the array R within the finite repetition are classified into the following two cases:

(a) Fig. 8 shows the discrete Markov graph in the transition of each pixel. Let the initial state of the array R be in the state S_r in Fig. 8. The new-state transition occurs by use of templates in the state S_t, while no state transition occurs in the state S_c. If there is no closed loop in the state S_t, the reslut R remains in either of the states S_r and S_c.

(b) The propagation of the signal is unidirectional. Namely, the array R is determined in order according to the specified direction.

As an example, let us consider the extraction of the connected components where the order of the chain is 1-2-3. The templates for the state transition are given below, where d denotes don't care.
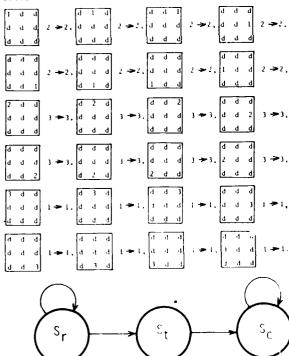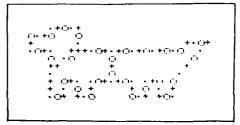
Fig. 8   State transition in the recursive near-neighbor instructions

The algorithm can be written by

R(Initial); given
Repeat R ← T(T(R, R, 2, R ; A), d, d, R ; P)
       where P ← NN(all 8) of T(3, 0, 3, 3 ; R)
       R ← T(T(R, R, R, 3 ; A), d, d, R ; P)
       where P ← NN(all 8) of T(3, 3, 0, 3 ; R)
       R ← T(T(R, 1, R, R ; A), d, d, R ; P)
       where P ← NN(all 8) of T(3, 3, 3, 0 ; R).
                                                    (19)

Fig. 9 shows the input image and the result based on these templates. After the state transition corresponding to the templates occurs, the center pixel remains unchaged. In other words, the state transition occurs once in the given templates. Therefore, the condition (a) is satisfied, and the result R is in the stable state after the finite repetition of the algorithm.

IV.   SIMPLIFICATION OF THE IMAGE PROCESSING ALGORITHM
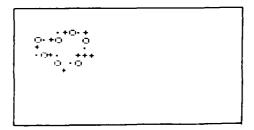
With the near-neighbor instructions, the number of templates required for the state transition often becomes large as shown in the previous example. In this case, the compression of the templates can be done using a minimization technique from multiple-valued logic [11, 12].

(a) Input image

(b) Initial state

(c) Final state

Fig. 9   Extraction of the connected components

Let the sum-of-products terms concerning templates be R_0, R_1, ..., R_{r-2} and R_{r-1}, where R_i is the sum-of-products term which causes the state transition to $i \in L$. The other terms which are not contained in the given templates can be written as

$$\overline{R_0} \cdot \overline{R_1} \cdot \ldots \cdot \overline{R_{r-1}} = \bigwedge_{i=0}^{r-1} \overline{R_i} \qquad (20)$$

The state transition in the center module does not occur, if the term of Eq. (20) takes the value r-1. Therefore, the state transition function $f(a_0, x_0, x_1, \ldots, x_8)$ is given by
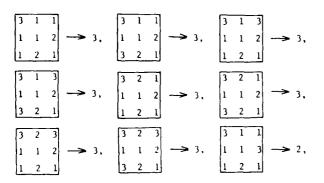
$$f(a_0, x_0, \ldots, x_8) = \bigvee_{i=1}^{r-1} i \cdot R_i + x_0 \cdot \bigwedge_{i=0}^{r-1} \overline{R_i}. \qquad (21)$$

In the operation in the MVAP, each sum-of-products term R_i is sequentially calculated, so that the minimum form of each R_i (i = 0, ..., r-1) is required for minimum execution time. From the above discussions, the procedure for obtaining the simplified image processing algorithm can be summarized as follows:

(Step 1) Find all the templates whose neighborhoods will produce a change of the state in the center pixel.
(Step 2) In order to get the minimum sum-of-products form in the function $g = \bigvee_{i=1}^{r-1} i \cdot R_i$, express the product term concerning each template using a cubical representation [12].
(Step 3) Obtain the minimum sum-of-products form in the function g from the cubes using the minimization technique [12].
(Step 4) Obtain the minimum sum-of-products form in the function R_0. In the ordinary minimization, this term is not necessary. However, the term R_0 is used as the the calculation of the function $x_0 \cdot \bigwedge_{i=0}^{r-1} \overline{R_i}$. A similar minimization technique can be applied to the term R_0.

Let the state transition function thus obtained be

$$f = \bigvee_{i=1}^{r-1} i \cdot S_i + x_0 \cdot \bigwedge_{i=0}^{r-1} \overline{S_i}. \qquad (22)$$

[Example] Consider a template matching whose state transition is given as follows:

$$\begin{bmatrix} 3 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 3, \quad \begin{bmatrix} 3 & 1 & 1 \\ 1 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} \longrightarrow 3, \quad \begin{bmatrix} 3 & 1 & 3 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 3,$$

$$\begin{bmatrix} 3 & 1 & 3 \\ 1 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} \longrightarrow 3, \quad \begin{bmatrix} 3 & 2 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 3, \quad \begin{bmatrix} 3 & 2 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 3,$$

$$\begin{bmatrix} 3 & 2 & 3 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 3, \quad \begin{bmatrix} 3 & 2 & 3 \\ 1 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} \longrightarrow 3, \quad \begin{bmatrix} 3 & 1 & 1 \\ 1 & 1 & 3 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 2,$$

$$\begin{bmatrix} 3 & 1 & 1 \\ 3 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 2, \quad \begin{bmatrix} 3 & 1 & 1 \\ 3 & 1 & 3 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 2, \quad \begin{bmatrix} 3 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 2,$$

$$\begin{bmatrix} 3 & 1 & 2 \\ 1 & 1 & 3 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 2, \quad \begin{bmatrix} 3 & 1 & 2 \\ 3 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 2, \quad \begin{bmatrix} 3 & 1 & 2 \\ 3 & 1 & 3 \\ 1 & 2 & 1 \end{bmatrix} \longrightarrow 2.$$

In this example, let all the patterns except these templates have a transition to 0. Therefore, the sum-of-products term of Eq. (21) is given by $f = \bigvee_{i=1}^{3} i \cdot R_i$. With respect to the terms R_3 and R_2, the following cubical representation can be obtained from the templates:

R_3

$\quad x_0 \quad\ x_1 \quad\ x_2 \quad\ x_3 \quad x_4 \quad\ x_5 \quad\ x_6 \quad\ x_7 \quad\ x_8$

(0100)(0001)(0100)(0100)(0010)(0100)(0010)(0100)(0100)

(0100)(0001)(0100)(0100)(0010)(0100)(0010)(0001)(0100)

(0100)(0001)(0100)(0001)(0010)(0100)(0010)(0100)(0100)

(0100)(0001)(0100)(0001)(0010)(0100)(0010)(0001)(0100)

(0100)(0001)(0010)(0100)(0010)(0100)(0010)(0100)(0100)

(0100)(0001)(0010)(0100)(0010)(0100)(0010)(0001)(0100)

(0100)(0001)(0010)(0001)(0010)(0100)(0010)(0100)(0100)

(0100)(0001)(0010)(0001)(0010)(0100)(0010)(0001)(0100)

R_2

(0100)(0001)(0100)(0100)(0001)(0100)(0010)(0100)(0100)

(0100)(0001)(0100)(0100)(0010)(0100)(0010)(0100)(0001)

(0100)(0001)(0100)(0100)(0001)(0100)(0010)(0100)(0001)

(0100)(0001)(0100)(0010)(0010)(0100)(0010)(0100)(0100)

(0100)(0001)(0100)(0010)(0001)(0100)(0010)(0100)(0100)

(0100)(0001)(0100)(0010)(0010)(0100)(0010)(0100)(0001)

(0100)(0001)(0100)(0010)(0001)(0100)(0010)(0100)(0001)

Using the minimization technique [12], we can obtain the following simplified terms:

S_3

(0100)(0001)(0110)(0101)(0010)(0100)(0010)(0101)(0100)

S_2

(0100)(0001)(0100)(0110)(0011)(0100)(0010)(0101)(0101)

The image processing algorithm becomes as

R3 ← P where P ← NN(5,8) of T(3,0,3,3;A)

R3 ← OR(R3,P) where P ← NN(4,6) of T(3,3,0,3;A)

R3 ← OR(R3,P) where P ← NN(1) of T(3,3,3,0;A)

R3 ← OR(R3,P) where P ← NN(2) of T(3,0,0,3;A)

R3 ← OR(R3,P) where P ← NN(3,7) of T(3,0,3,0;A)

R3 ← T(0,T(3,d,d,0;R3),0,0;A)

R2 ← P where P ← NN(2,5,7) of T(3,0,3,3;A)

R2 ← OR(R2,P) where P ← NN(6) of T(3,3,0,3;A)

R2 ← OR(R2,P) where P ← NN(1) of T(3,3,3,0;A)

R2 ← OR(R2,P) where P ← NN(3) of T(3,0,0,3;A)

R2 ← OR(R2,P) where P ← NN(4) of T(3,3,0,0;A)

R2 ← OR(R2,P) where P ← NN(8) of T(3,0,3,0;A)

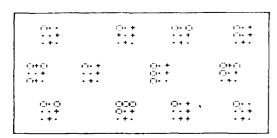· · ← T(0,T(2,d,d,A;R2),0,0;A)

R ← OR(R3,R2).

It can be seen that the execution steps can be greatly reduced by the simplification. Fig. 10 shows the example of this image processing.
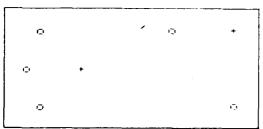
V. CONCLUSION

In this paper, the image processing algorithms for the MVAP have been presented. The simplification of the image processing algorithms is discussed using the minimization technique of sum-of-product forms in the multiple-valued logic. By the method developed, various image processings can be performed in a short time. The simplification is closely related to the compression of templates, so that the concept can applied to feature extraction in pattern recognition.

As a future problem, we need to consider the implementation of MVAP using multiple-valued elements. One of the most important problems in the implementation is the use of low-cost multiple-valued memory. At the present time, CCD with level refreshers is the most promising technology for use in multiple-valued memory.

REFERENCES

[1] A. Rosenfield, Picture Processing by Computer, Academic Press, New York, 1969.

[2] A. Rosenfield and A. C. Kak, Digital Picture Processing, Academic Press, New York, 1976.

[3] S. H. Unger, "A Computer Oriented toward Spatial Problems," Proc. IRE, pp. 1744-1750, Oct. 1958.

[4] B. Kruse, "A Parallel Picture Processing Machine," IEEE Trans. Comput., C-22, pp. 1075 -1087, Dec. 1973.

[5] A. P. Reeves, "A Systematically Designed Binary Array Processor," IEEE Trans. Comput., C-29, pp. 278-287, April 1980.

[6] D. C. Rine, "Picture Processing Using Multiple-Valued Logic," Proc. Eleventh International Symposium on MVL, pp. 73-78, May 1981.

[7] D. C. Rine, "Associative and Multi-Valued Logic for Possible Improvements in Some X-Ray Image Processing," Proc. Fifth International Symposium on MVL, pp. 146-161, May 1975.

[8] M. Kameyama and T. Higuchi, "A New Digital Image Processor Using Multiple-Valued Logic," Proc. Twelfth International Symposium on MVL, pp. 8-16, May 1982.

[9] T. Higuchi and M. Kameyama, "Ternary Logic System Based on T-Gate," Proc. Fifth International Symposium on MVL, pp. 290-304, May 1975.

[10] M. Kameyama and T. Higuchi, "Synthesis of Optimal T-Gate Networks in Multiple-Valued Logic," Proc. Ninth International Symposium on MVL, May 1979.

[11] C. M. Allen and D. D. Givone, "A Minimization Technique for Multiple-Valued Logic System," IEEE Trans. Comput., C-17, pp. 182-184, 1968.

[12] Y. H. Su and P. T. Cheung, "Computer Minimization of Multi-Valued Switching Functions," IEEE Trans. Comput., C-21, pp. 995-1003, Sept. 1972.

(a) Input image

(b) Pattern matching

Fig. 10   Example of simple near-neighbor instructions

TERNARY TRANSMISSION IN LOCAL AREA NETWORKS

S. G. Zaky and Z. G. Vranesic

Department of Electrical Engineering
University of Toronto

ABSTRACT

This paper proposes a number of ternary codes
for the transmission of binary data in local area
networks. The proposed codes are suitable for
asynchronous transmission. It is very easy for the
receiver to recover the transmission clock, without
the need for phase-locked loops and related hard-
ware. Provision is also made for unique represen-
tation for message delimiters, or flags. The main
advantage of the proposed approach is the simplifi-
cation of the transmitter and the receiver hardware.

1. INTRODUCTION

Multivalued logic has been the subject of consi-
derable research activity during the past two
decades. Many techniques for design of multivalued
switching functions have been proposed, and a
number of potentially useful electronic circuits
for their implementation have been developed. Less
successful have been attempts to apply multivalued
schemes in engineering practice. There are several
reasons for this, perhaps the main one being the
fact that binary technology is well understood and
readily available. A multivalued alternative is
likely to be tried only if it offers some clear
advantage over the binary case.

The evolution of multivalued logic techniques
has not reached a point where large systems may be
built using multivalued logic exclusively. However,
there are interesting possibilities where multi-
valued approaches can be used to advantage within
what are essentially binary systems. This paper
pursues one such possibility, namely the use of
ternary signalling in a ring-structured local area
network.

It has been advocated for some time that multi-
valued logic provides natural means for reducing
interconnection complexity [1,2]. The most obvious
application would involve using binary integrated
circuits with multivalued signals on the input and
output pins of the IC packages. Each IC would have
to include the necessary decoder and encoder cir-
cuits to translate the incoming multivalued signals
to binary and the outgoing signals back to multi-
valued. All of the processing within the chips
would involve standard binary circuits. While this
approach may be attractive and generally applicable,
no IC's of this type have been developed
commercially.

A more plausible application in the near future
may be to exploit the reduced interconnection
complexity at a subsystem level, particularly for
interconnecting subsystem units. This may be
considered with systems that involve relatively
long interconnection links, as in the case of
local area networks. Here, binary subsystem
units, namely the receiver/transmitter stations,
might be advantageously interconnected through
multivalued channels. This is a possibility that
we think should be investigated.

Section 2 of this paper discusses the relevant
aspects of local area networks, focussing on the
ring structure. The next section deals with the
possible ways of encoding the transmitted data.
Section 4 considers the transmission issues, and
presents a practical scheme for implementing
ternary transmission. Finally, the results
obtained from prototype circuits are described.

2. RING STRUCTURED LOCAL AREA NETWORKS

A local area network (LAN) provides communica-
tion links between digital equipment spread over a
geographical area that may span distances of up to
a few kilometers, but more typically up to a few
hundred meters. A large building wired for this
purpose is a good example of the size that a LAN
may be expected to have. Three types of LANs are
of practical significance: bus, ring and star
structured networks.

Bus LANs have gained an early acceptance,
mainly due to the commercial availability and
popularity of Ethernet [3]. They are highly suit-
able for traffic dominated by large file transfers,
but not very appropriate for character based
traffic. Ring LANs offer an alternative that
allows smooth handling of both file and character
based traffic. Their critics claim that this
advantage is offset by the necessity for a more
rigid physical configuration and difficulties in
maintaining partial operation of the network when
some parts of it fail. Star LANs make use of the
well understood telephone technology, which is
their chief advantage. On the other hand they
require the most extensive wiring plant. An
illuminating discussion of the relative merits of
the three types of LANs can be found in
reference [4].

242

Our objective is to scrutinize the problems related to the transmission techniques used in LANs. In particular, we are interested in using multivalued logic to simplify the transmission protocol, and hence the receiver design. Multivalued implementations are attractive in point-to-point wiring, which is the case in ring and star LANs. Our preference is for ring LANs, hence this is the structure that we have concentrated on.

A number of ring LANs have been constructed [5-8], but they have not yet proliferated in practice on the scale of bus LANs. However, the employment of LANs is still a recent phenomenon. A true assessment of the popularity of any given structure will only be possible in a few years time, when experiences with numerous LANs are accumulated. It is interesting to note that the foremost computer manufacturer, the IBM Company, recently announced the choice of the ring as the structure for its LAN [9].

Transmission related considerations in the design of ring LANs include the following:

- speed,
- transmission medium,
- synchronization,
- transmission code,
- complexity of the transmitter and the receiver,
- supply of power, and
- fault tolerance.

A typical ring LAN is depicted in Figure 1. Each device is connected to the ring by means of a station, which contains the required transmitter and receiver circuits, as well as an appropriate interface to the device. The transmission medium depends upon the speed requirements. In very high speed applications, in the range of 50-100 MHz, the medium may be coaxial cable or optical fibres. In the range of 1-10 MHz, it is possible to use ordinary twisted-pair wire. Thus, the cost of cabling is largely dependent upon the cost of labor involved in its installation.

Most ring LANs are likely to operate in environments where high speed of transmission is not a primary requirement. In fact, baseband transmission under 10 MHz is quite adequate. Thus, utilization of the available bandwidth need not be an overriding concern. The important issues are the cost and reliability of operation.

The cost of a LAN is a major factor. It has greatly influenced the design of two of the above mentioned LANs [5,8]. The ultimate goal is to have simple enough stations so that they can be implemented as single IC chips. Thus, the complexity of the transmitting and receiving circuits must be kept to a minimum.

The transmission code is of utmost importance. Data is transferred in packets, delimited by flags. The code used must allow for easy designation of the transmitted data, flags and any control information that may be required. It is particularly useful if flags can be asserted as unique code patterns. This simplifies message synchronization and recovery from failure. Such an arrangement is difficult to achieve with presently available binary schemes. For example, the extensively used Manchester code defines uniquely only two symbols, 0 and 1. Unique flags can be represented only as violations of the code.

Another requirement may be that the transmission code be electrically balanced. This is necessary in transformer coupled systems.

Clock recovery is a key consideration. Correct interpretation of the received signal depends upon the ability to extract the clock information from the transmitted data. Present systems inevitably employ a phase locked loop in each station for this purpose. Phase locked loops provide a reliable means for clock recovery during normal operation of a LAN. However, synchronization is lost if transmission on the LAN stops, either because of temporary failures or at times when stations are inserted or removed from the LAN. In order to provide for fast resynchronization, the phase locked loop is augmented with additional circuitry. An interesting discussion of the related problems and possible solutions if found in Muller at al [10]. Our objective is to study transmission codes which allow simpler recovery of the clock in order to reduce the complexity of the receiving circuits in a station.

3. TRANSMISSION CODE

The discussion in the previous section suggested that the transmission code used in a LAN should have two important characteristics. First, it should provide a simple mechanism for transmission clock recovery. Secondly, it should allow easy separation of data and control information by having unique and easily recognizable codes for one or more flags. This means that the transmission code should be capable of representing at least three values. For example, values 0 and 1 may represent the transmitted information, while the third value serves as a flag F. High utilization of the available bandwidth, while important, should not be achieved at the expense of increased complexity of the transmission hardware.

Consider a transmission scheme which allows S distinct symbols, one of which is transmitted during each clock period. The maximum information carrying capacity of such a link is represented by the case where one of S different values, or digits, is transmitted in each clock period. Such is the case for binary NRZ codes [11]. We will use this as a basis for evaluating code overhead for different transmission codes. Thus, if a particular code allows one of V digits, where $V \leq S$, to be transmitted every period, the overhead will be taken as $(S-V)/S$.

We will examine transmission codes in which a transition is always present at the boundary between any two successive symbols and nowhere else. This makes it easy for the receiver to recover timing information. Such codes lend themselves to completely asynchronous transmission, where the duration of each transmitted symbol is independent of the duration of earlier symbols. Furthermore, the receiver can clock in each symbol without

243

having precise knowledge about the transmission clock period.

Asynchronous codes for binary transmission were studied by M'Rabet et at [12]. The code proposed by M'Rabet involves more than one transition per symbol, which is a necessity in the binary case. As will be shown below, a much simpler receiver can be realized if multiple-valued transmission is used.

In the proposed ternary code, no two successive symbols can be identical. For a given symbol in period i-1, only one of the remaining S-1 symbols can be transmitted during period i. As a result, the minimum code overhead is 1/S. This implies that high utilization of the bandwidth can be achieved only with a large value for S.

Consider a ternary, 4-wire transmission system. Since 2 signals are transmitted, each having 3 possible values, a total of 9 code symbols exist. In view of the self-clocking constraint, only 8 symbols are available for transmission. Thus, it is possible to transmit the equivalent of 3 bits of data in one clock period. However, this would not leave code space for a unique flag. A more interesting alternative is to provide for transmission of either 2 bits of data or one of 4 flags in a given clock period. This may be achieved with the code shown in Figure 2. Note that each digit is uniquely defined in terms of the current symbol i and the symbol i-1 transmitted in the previous clock period.

A similar code may be derived for binary transmission on a 4-wire facility. In this case, 4 transmission symbols are available, which means that only the 3 values 0, 1 and F can be encoded. Such a code is given in Figure 3. A code of this type, but without a provision for a unique flag is used in the Cambridge Ring [5].

Finally we should note that a ternary, 4-wire scheme allows other possibilities. In any 4-wire system consisting of 2-wire pairs, there is a potential difficulty caused by signal skew, i.e. by differences in the propagation delay along the two separate transmission paths. The effect of skew can be eliminated if the code is designed such that a signal transition must occur on each of the two pairs for each symbol transmitted. The receiver then simply waits for both transitions to arrive, before recognizing a new symbol. Obviously, this approach reduces the number of digits that can be transmitted. An example of such a code is given in Figure 4.

In the following section, we will show that a simple receiver design can be used for any of the codes of Figures 2, 3 and 4. It performs both the decoder and clock recovery functions.

4. CLOCK RECOVERY

The codes presented in section 3 guarantee the presence of one transition between any two transmitted symbols. Thus, the transmission clock may be recovered by a transition sensitive device, or a differentiator.

A simple circuit which uses this approach is given in Figure 5. Two ternary receivers generate 4 binary signals Y_{1-4}. These signals are connected to a 4-bit input latch as well as to 4 address lines of 256 x 4 bit read only memory. The outputs of the latch are connected to the other 4 address lines. The least significant bit in the ROM is programmed so that it contains a 0 whenever the low and high order nibbles of the address are identical, and it contains a 1 elsewhere. Starting with a random state in which the received information is different from that stored in the latch, the clock output of the ROM will be in the 1 state. Thus, the received data will be loaded into the latch, making the high and low order nibbles of the address identical. As a result, the clock output becomes 0. This sequence of events will be repeated whenever a new transmission symbol is received [13].

The block labelled Δ represents an integrating delay, intended to filter out any multiple pulses which may appear at the ROM output during address decoding. Also, it compensates for the effect of skew between the two ternary signals.

One of the 4-wire codes presented in the previous section, namely that of Figure 4, is characterized by the presence of a transition in each of the two ternary signals between any two successive symbols. The receiver realization should take advantage of this feature to compensate for transmission skew. In the scheme of Figure 5 this can be easily implemented by suitable choice of the bit pattern stored in the ROM. A 1 should be stored in those ROM locations whose addresses are consistent with this constraint of the code.

Only one bit of each word in the ROM is used for clock recovery. The remaining 3 bits can be used for storing the decoding table for the transmission code. The receiver arrangement shown allows this code to be based on both the current and previous transmission symbols. Hence, any of the codes discussed in the previous section can be supported.

5. TRANSMISSION

A few schemes have been proposed for ternary baseband transmission. A simple scheme is to introduce an intermediate voltage level to standard TTL level [14]. This approach is suitable only for very short distances, such as encountered in computer backplane interconnections. Another 3-valued scheme [15] has been suggested for transmission over distances of up to 1.5 km. However, the complexity of the receiver required makes it unsuitable for use in a LAN environment. As pointed out earlier, simplicity and low cost are the key requirements in the design of LAN hardware.

We are presently testing a differential transmission scheme, illustrated in Figure 6. It is based on the use of standard binary drivers and receivers. The differential signal, $Z_1 - Z_2$, obtained from two single-ended line drivers is transmitted over a twisted pair. Thus, the

transmitted signal can be regarded as having three states 0, 1 and 2, which correspond to X_1X_2 = 01, 00 and 10, respectively.

A ternary differential receiver is implemented in the form of two optical isolators connected in opposite directions, as shown in the figure. It can be easily verified that, in the absence of transmission errors, the two outputs Y_1 and Y_2 are equal to X_1 and X_2, respectively.

The system shown in Figure 6 has been implemented using DS8831 TTL line drivers and GN137 optical couplers. It has been tested with a transmission line consisting of 1000 feet of ordinary telephone cable at a baud rate of 5 MHz. The differential signal at the input of the receiver is shown in Figure 7.

Because of the finite bandwidth of the transmission channel, different level transitions will result in different delays. This is illustrated in Figure 8. The delay introduced for a given transition is a function of the ratio of the threshold voltage V_t to the maximum voltage V.

Assuming exponential behaviour with a time constant τ, the four delays defined in the figure may be estimated as follows

$$b_{21} = \tau \ln (V/V_t) ,$$
$$b_{10} = \tau \ln (V/(V - V_t)) ,$$
$$b'_{02} = \tau \ln (2V/(V + V_t)) , \text{ and}$$
$$b_{02} = \tau \ln (2V/(V - V_t)) .$$

For example, if V_t = 0.5V, we obtain

$$b_{21} = b_{10} = 0.69 \tau ,$$
$$b'_{02} = 0.29 \tau, \text{ and } b_{02} = 1.39 \tau .$$

The extent of the variations in the transmission delay on the 5 MHz experimental link tested by the authors is illustrated in Figure 9. This waveform is in fact a composite obtained by superimposing the 4 receiver outputs Y_1 to Y_4 while random data was being transmitted. The width of the transition region is about 100 ns.

Variations in delay have two effects. These are:

1) the introduction of signal skew within the receiver, and
2) the introduction of jitter.

Signal skew is perhaps the most serious problem for which allowance must be made in the receiver circuit. Because of the presence of skew, a clock pulse may be generated by the clock recovery circuit before all data outputs have settled. The delay Δ in Figure 5 is intended for this purpose. If system parameters are chosen such that V_t/V is in the range 0.25 to 0.5, the channel delay for all possible transitions will be in the range 0.29 τ to 1.39 τ. Hence, the deskewing delay Δ should be set to 1.5 τ as a minimum.

Because of the completely asynchronous nature of the receiver, jitter has no effect, provided

that the duration of each symbol is long enough to be recognized. When a number of links are operated in tandem, as in a ring network, a problem can arise. Jitter causes the duration of successive symbols to be unequal, and the effect can be cummulative over several links. Therefore, it is important to ensure that the duration of any symbol does not drop below some minimum value. This may be accomplished by delaying the response of the receiver to a transition if this transition occurs before the required minimum duration for a symbol. The delay block labelled Δ in Figure 5 can be easily designed to implement this feature. The circuit used by the authors is given in Figure 10.

6. CONCULDING REMARKS

This paper has investigated the possibility of using a ternary scheme for asynchronous transmission of binary data. Several possible codes have been suggested. These codes are intended to provide self-clocking and lead to simple transmitter/receiver circuits. They also provide one or more unique representations for flags.

We are presently testing prototype circuits for implementation of the proposed scheme. The basic transmission and clock recovery circuits have been tested successfully. The experimental results indicated that the designed circuits provide reliable operation up to a baud rate of 5 MHz.

7. REFERENCES

[1] E.J. McCluskey, "A Discussion of Multiple-valued Logic Circuits," Proc. 12th ISMVL, Paris, France, May 1982, pp. 200-205.

[2] Z.G. Vranesic, "Applications and Scope of Multiple-valued LSI Technology," Proc. COMPCON Spring 81, San Francisco, Feb. 1981, pp. 213-216.

[3] "The Ethernet," Version 1.0, Digital Equipment Corp., Intel Corp., and Xerox Corp., Sept. 1980.

[4] J.H. Saltzer, D.D. Clark and K.T. Pogran, "Why a Ring?", Proc. Seventh Data Comm. Symp., Mexico City, Oct. 1981, pp. 211-217.

[5] M.V. Wilkes and D.J. Wheeler, "The Cambridge Digital Communication Ring," Proc. Local Area Comm. Network Symp., May 1979, pp. 47-61.

[6] J.H. Saltzer and K. Pogran, "A Star-Shaped Ring Network with High Maintainability," Proc. Local Area Comm. Network Symp., Mitre Corp., May 1979, pp. 179-190.

[7] E.R. Hafner, Z. Nenadal and M. Tschanz, "A Digital Loop Communication System," IEEE Trans. on Comm., Vol. 22, June 1974, pp. 877-881.

[8] Z.G. Vranesic, V.C. Hamacher, W.M. Loucks and S.G. Zaky, "TORNET: A Local Area Network," Proc. Seventh Data Communications Symp., Mexico City, Oct. 1981, pp. 180-187.

[9] D.W. Andrews and G.D. Schultz, "A Token-Ring Architecture for Local Area Networks:

An Update, "Proc, COMPCON 82, Sept. 1982, pp. 615-624.

[10] H.R. Müller, H. Keller and H. Meyr, "Transmission in a Synchronous Token Ring," IBM Research Report No. RZ1122, Zürich, Dec. 1981.

[11] R.H. Severt, "Encoding Schemes Support High Density Digital Data Recording," Computer Design, Vol. 19, No. 5, May 1980, pp. 181-190.

[12] N. M'Rabet, G. Noguez and D. Trecourt, "Reseaux Locaux a Tres Haut Debit: L'Equivalent du Code de Transmission Asynchrone 'Start-Stop'," Proc. Euromicro, 1981.

[13] Z.G. Vranesic and S.G. Zaky, "Multivalued Logic in Local Digital Interconnection System," Proc. COMPCON 82, Sept. 1982, pp. 127-134.

[14] D. Etiemble, "Multivalued Integrated Circuits for Microcomputer Systems," Proc. COMPCON 82, Sept. 1982, pp. 135-144.

[15] C.W. Ross, "Reducing System Interconnections with Multivalued Logic," Electronics, Sept. 1977, pp. 122-124.

Symbol i-1	Symbol i							
	Transmitted Digit							
	00	01	10	11	F1	F2	F3	F4
00	01	02	22	21	11	12	10	20
01	02	10	00	22	12	20	11	21
02	10	11	01	00	20	21	12	22
10	11	12	02	01	21	22	20	00
11	12	20	10	02	22	00	21	01
12	20	21	11	10	00	01	22	02
20	21	22	12	11	01	02	00	10
21	22	00	20	12	02	10	01	11
22	00	01	21	20	10	11	02	12

Figure 2. An 8-valued code for ternary, 4-wire transmission.
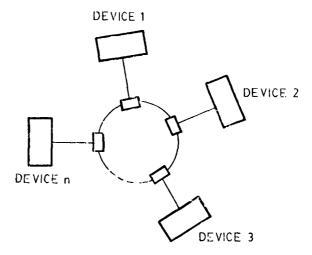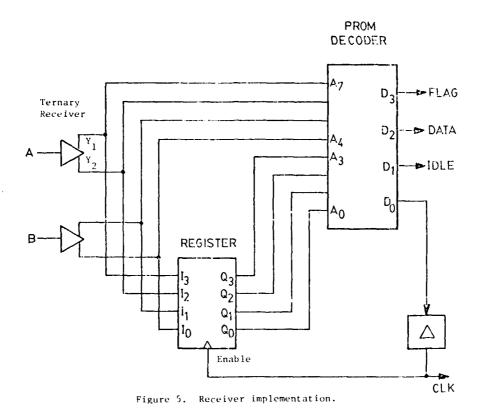
Figure 1. A ring-structured local area network.

Symbol i-1	Symbol i		
	Transmitted Digit		
	0	1	F
00	01	10	11
01	10	11	00
10	11	00	01
11	00	01	10

Figure 3. A 3-valued code for binary, 4-wire transmission.

Symbol i-1	Symbol i			
	Transmitted Digit			
	0	1	F1	F2
00	11	12	21	22
01	12	10	22	20
02	10	11	20	21
10	22	21	02	01
11	20	22	00	02
12	21	20	01	00
20	01	02	11	12
21	02	00	12	10
22	00	01	10	11

Figure 4. A 4-valued code for ternary, 4-wire transmission.

Figure 5. Receiver implementation.
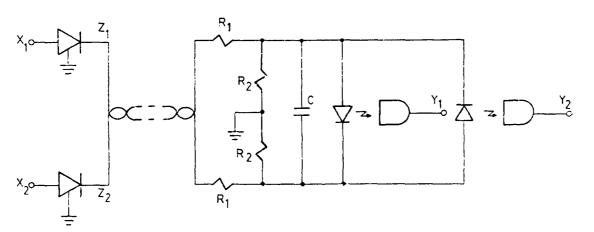
Figure 6. Differential ternary transmission using single-ended drivers
and optical isolators.
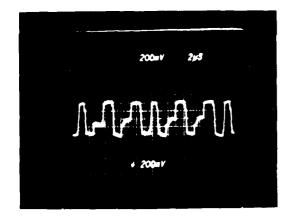
Figure 7. Waveform at the input of the receiver
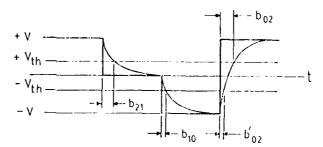
Figure 8. Delay for defferent transitions.
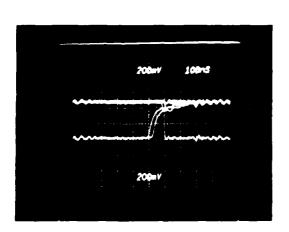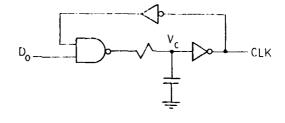
Figure 9. Signal skew.
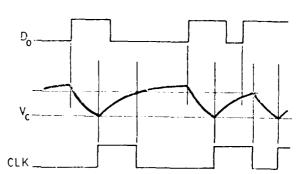
Figure 10. Circuit for duty cycle restoration.

SOME DEVICE COUNT COMPARISONS FOR REDUCED CONTROL STORES USING

MULTIPLE-VALUED MOS CIRCUITS

Charles B. Silio, Jr. and James H. Pugsley

Electrical Engineering Department
University of Maryland, College Park, MD 20742

ABSTRACT

The design of decoders for multiple-valued MOS
read-only memories (ROMs) used for reducing chip
area in microprogrammed digital processors is con-
sidered. Using the threshold detection circuitry
implemented by Intel Corporation, we present
designs for one-out-of-four, one-out-of-sixteen,
and one-out-of-thirty-two decoders for use with
optimal radix four encodings of the microopera-
tions in control stores, thus providing single
layer decoding of the microoperations at the ROM
outputs. We also include discussion of a design
for a one-out-of-eight decoder for octal ROM
cells. We use a newly obtained radix four optimal
grouping of microoperations for the 256 word, 75
microoperation control store example derived from
the control store of a Digital Equipment Corpor-
ation PDP-11/40 central processing unit to
illustrate device counts obtainable, and we com-
pare these counts to the two layer decoding scheme
obtained by using a k-bits per cell encoding
with 2^k-to-binary code translation.

I. Introduction

Requirements for increased functionality and
speed for single-chip integrated circuit (IC)
microprocessors, coupled with those for ever wider
data buses for on-chip information flow, lead to
increasing IC chip areas with corresponding impact
on yield. Multiple-valued circuits used in con-
junction with existing binary circuit components
provide a way for realizing increased func-
tionality without suffering corresponding
decreases in IC yield under both present and
future processing technologies. Even if problems
associated with reducing feature sizes, such as
those associated with the scaling of metal lines
[1], are successfully solved, problems of
increased functionality versus available chip real
estate and yield will remain. Greater func-
tionality and orderly processor implementation
techniques will continue the present trend
requiring placement of ever larger control stores
or read-only memory arrays on single-chip
microprogrammed processors. Because not all
control points in a microprogrammed processor can
be meaningfully activated at the same time,
various forms of encoded field control are used to
encode subsets of microoperations into fields in
order to reduce the width of ROM words at the
expense of introducing additional circuitry with
corresponding delays to decode the encoded fields.
Two forms of encoded field control are of interest
here. First are optimal and suboptimal directly
(or minimally) encoded formats that employ a

single layer of ROM output decoding; and second are
the k-bits per cell formats that use r-valued ROM
cells with $r=2^k$ and that employ 2^k-to-binary code
translators as a second layer of ROM output
decoding inserted between the ROM outputs and the
binary decoders used for binary direct encoding.

It is now well established that reduced micro-
instruction word width achieved through multiple-
valued encoding of microoperations and implemented
using multiple-valued circuits in the form of ROMs,
code translators, and decoders significantly redu-
ces chip area for the control store portion of
single-chip processors. The use of quaternary HMOS
ROM cells and quaternary to binary code translators
in the INTEL 8087 Numeric Data Processor [2],[3],
and the iAPX 43203 Input/Output processor [3],[4],
demonstrate that at least with four-valued circuits
the gain in yield through IC chip area reduction
overcomes the disadvantages of meticulous pro-
cessing needed to construct reliable and reprodu-
cible multiple-valued circuits.

A further discussion of the effects on IC yield
resulting from choice of either binary or quater-
nary ROM circuits in the two-bits per cell form is
presented in section II.

The main thrust of this presentation is a con-
tinuation of a discussion begun in Silio et al.[5];
however, here we make use of metal-oxide semicon-
ductor (MOS) field effect transistors. Optimal and
alternative suboptimal encodings of microoperations
are often available for direct encoding and storage
in a quaternary (or even octal) ROM leading to the
speed advantage of a single layer of output
decoding over the two layers usually required for
code translation and decoding in a two (or three)
bits per cell scheme. To make use of this speed
advantage, one-out-of-four (or one-out-of-8), one-
out-of-sixteen, and so on through one-out-of-4^m
decoders of practical complexity are required to
decode the directly encoded microoperation infor-
mation on the quaternary digit lines of the ROM
array. In section III we present designs for
several such output circuits that make use of input
signals from Intel's latching differential
threshold detector (LDTD) circuit (Fig. 12 in
Bayliss et al.[4]), thus expanding the utility of
those basic and successful multiple-valued cir-
cuits.

In section IV we reconsider the 256 word, 75
microoperation control ROM specification derived
from the actual ROM specification [6] for the
Digital Equipment Corporation PDP-11/40 computer.
This microcode specification was presented in Silio
et al.[5], and for economy of space is not repro-
duced here. For this example the prediction of a

lower bound digit dimension using the tools derived in Jeng [7] & [8] and presented in Silio et al.[9] & [10] results in a lower bound microinstruction word width of $D_{\ell b}=15$ for micro-instructions that are directly encoded using quaternary digits for storage in four-valued ROM cells.

In Silio et al.[5] we presented a suboptimal grouping of the microoperations to be encoded into quaternary ROM words that require 16 digit positions. The branch and bound depth first search algorithm of Baer [11], modified to search for multiple-valued encodings as shown in Silio et al. [12], was used to find an optimal radix $r=4$ grouping of the microoperations for encoding in 15 digit wide quaternary ROM words. This example is used to motivate the choice of decoder designs presented in section III.

We then compare device counts required for implementing in MOS circuits this optimal $r=4$, digit dimension $D=15$ direct encoding of micro-operations to those needed in a more easily found $D=16$ suboptimal encoding presented previously [5], as well as to a suboptimal two-bits per cell encoding in this empirical example to see if there are significant differences in the approaches.

Because an optimal radix $r=8$ direct encoding of the microoperations in the PDP-11/40 based example has also been found, and because in principle there is no obstacle to implementation of MOS eight-valued ROM cells in a fashion similar to that for quaternary cells, we include a design for a one-out-of-eight decoder in section III and use it in our device count comparisons in section IV.

II. Estimates of Yield Variation

The yield Y of working IC chips per wafer processed has been described by the equation $Y=ke^{-FA}$ [13] & [14], where k is a proportionality constant, F is the defect density (i.e., flaws per unit area) on the processed wafer and A is the chip (or die) area. Whitney [14] cites defect densities of F equal 10 to 20 per square centimeter (cm^2) as typical of early 1980's processing.

Given one IC chip with area A_1 and another IC chip with area A_2 greater than A_1, one can then estimate the percent reduction in yield ($Y_{RED.}$) due to this increased area, assuming other processing variables to be constant (viz., $k_1=k_2=k$

and $F_1=F_2=F$). For $Y_1=ke^{-FA_1}$ and $Y_2=ke^{-FA_2}$, we find that $Y_{RED.}=[(Y_1-Y_2)/Y_1]X100\%=[1-e^{F(A_1-A_2)}]X$ 100%.

According to Nave [15], the IC chip for the Intel 8087 represents an "area larger than 280 mils square." Although the die for this processor is rectangular, we can estimate the deleterious effect on yield caused by increased area by assuming $A_1=280X280$ $mils^2=78.4K$ $mils^2=0.5058$ cm^2.

Stark [3] points out that if the 8087 would have been implemented using a standard binary ROM instead of the quaternary two-bits per cell ROM actually used, its area would have been 8% larger; hence, $A_2=1.08A_1$. Estimating the decrease in yield suffered by using a larger area binary control store, we see that if $F=10$ per cm^2, then

$Y_{RED.}=33.3\%$; and if $F=20$ per cm^2 then $Y_{RED.}=55.5\%$; rather significant decreases.

Stark [3] also states that use of a two-bits per cell ROM on the iAPX 43203 resulted in a 31% decrease in control store chip area over a purely binary implementation. From information in [4] and measurements of a photograph in [16] it appears that the microinstruction ROM on this chip represents about 10.2% of overall area. Die area for this chip is $A_1=326X358$ $mils^2=116708$ $mils^2=0.753$ cm^2 of which approximately 11886 $mils^2$ would be devoted to the control store. With this information we estimate the die area resulting from use of a purely binary ROM as $A_2=(1.0458)A_1$, which corresponds to a 4.6% overall increase. If F equals 10 per cm^2, then the estimated reduction in yield would be $Y_{RED.}=29.1\%$; and if F equals 20 per cm^2, then $Y_{RED.}=49.8\%$. The advantage of using quaternary instead of binary ROM cells is clearly increased yield, even when overhead devices in the quaternary-to-binary code translators are taken into account.

In the next section we consider design of decoders in addition to Intel's quaternary-to-binary code translator in order to enhance opportunities for using MOS quaternary ROM cells in situations when greater chip area reduction over that provided by a two-bit per cell structure might be obtainable from optimal and alternative suboptimal microoperation encoding schemes.

III. Some New Decoder Designs

Intel's four-valued MOS ROM cell [3] uses a transistor having one of four distinct channel widths and, hence, one of four possible resistance values to store a quaternary value at digit position j of word i in the ROM. In order to discriminate which value is stored at that ij^{th} location, three effectively simultaneous actions are performed.

First, word i is selected by the address decoding circuitry to connect the ij^{th} resistance value between the jth digit line and ground while all other j^{th} digit line devices act as open circuits.

Second, a feedback compensated sense current is driven down the jth digit line to generate sense voltage V_S on this line using the ROM cell transistor as part of a voltage divider circuit. At the same time three threshold reference voltages V_{R1} V_{R2} V_{R3} are generated by identical drive circuitry using each of three reference transistors with resistances R1 R2 R3 to form three independent voltage divider circuits. The reference devices are adjusted so that their impedances lie strictly between the impedance values chosen for the ROM cell devices. V_S is then threshold detected against each of the reference voltages V_{R1}, V_{R2}, and V_{R3}, respectively, using three differential amplifiers with binary logic outputs x_1, x_2, and x_3. If V_S is less than V_{R1}, then $x_1=x_2=x_3=0$. If V_S is greater than V_{R2} but less than V_{R3}, then $x_1=x_2=1$ and $x_3=0$, and so on for all four possible combinations of threshold values in the set x_t, $t=1,2,3$. An improved threshold detection circuit reported in Bayliss [4] combines the differential amplifier for comparing V_S and V_{Rt} with a latch to generate the output signal x_t and its

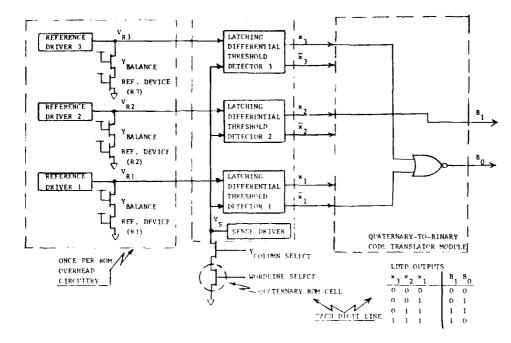Table 1: One-out-of-8 decode functions for 8-valued ROMs and LDTD outputs $x_1, \cdots, x_7$.

$f_0 = \overline{x}_1$	
$f_1 = x_2 + \overline{x}_1$	
$f_2 = x_3 + \overline{x}_2$	
$f_3 = x_4 + \overline{x}_3$	
$f_4 = \overline{x}_4 + x_5$	
$f_5 = \overline{x}_6 + \overline{x}_5$	
$f_6 = x_7 + \overline{x}_6$	
$f_7 = x_7$	

LDTD OUTPUTS

x_3	x_2	x_1	B_1	B_0
0	0	0	0	0
0	0	1	0	1
0	1	1	1	1
1	1	1	1	0

Fig. 1: Quaternary to binary translator for 4-valued ROM with circuits for reference voltage generation, threshold detection (LDTDs), and binary signals $B_1 B_0$ with $B_1 = x_2$ and $B_0 = \overline{x}_3 x_1$.

logical complement $\overline{x}_t$. Three of these latching differential threshold detectors (or LDTDs for short) from [4] are shown in Figure 1, and it is this LDTD circuit that we assume for use in the following discussion. Note that in general if r-valued ROM cell circuits can be reliably constructed, then r-1 reference voltages V_{R1} V_{R2} $\cdots V_{R(r-1)}$ must be generated, and V_S must be threshold detected against each V_{Rt} using r-1 such LDTD modules that generate the r-1 pairs of binary switching values x_t and $\overline{x}_t$, for $t=1,2,\cdots, (r-1)$. We have included Figure 1 here to document our choice of subscript notation for both the reference voltages and the corresponding x_t logical values, as well as to show the location of the binary switching circuits that replace the rightmost output module in Figure 1, labeled "quaternary-to-binary code translator module".

This then is the third action to be performed, namely, the generation of the appropriate binary output signals that provide 2^k-to-binary code translation for the k-bits per cell encoded micro-operations. The outputs of this switching circuit module could also generate appropriate microoperation control signals for r-valued direct encodings of the microoperations. These binary switching modules have as input the x_t's from the $t=1,2,\cdots$, (r-1) LDTDs and generate the desired logical outputs. The resulting decoder designs presented below enhance and extend the utility of the multi-valued ROMs, the V_{Rt} reference circuits, and the LDTDs.

Each reference driver from [3] & [4] comprises 3 devices configured as 2 depletion mode devices and 1 enhancement mode device. The sense drivers

attached to each and every digit line are identical to the reference drivers and have the same device counts. Counting the 2 enhancement mode devices comprising the reference transistor (R_t) and the series $Y_{BALANCE}$ transistor inserted to correspond to the coincident selection $Y_{COLUMN-SELECT}$ transistor attached to each and every digit line, we see that the reference circuit needed to generate V_{Rt} comprises a total of 5 devices configured as 3 enhancement mode plus 2 depletion mode devices, for $t=1,2,\cdots, (r-1)$. Each LDTD in [4] comprises 11 devices configured as 5 depletion mode and 6 enhancement mode devices. NOR gates such as that appearing in the binary switching module are composed of 1 depletion mode pull-up plus 1 enhancement mode device for each gate input. The quaternary circuit in Figure 1 thus requires a total of $(3+33+3)=39$ devices per digit line (versus 43 for the configuration in Stark [3]) to sense, threshold detect, and then translate to binary the contents of the selected quaternary ROM cell, plus a once per ROM constant overhead of 15 devices in the reference set.

An r=4 one-out-of-four digit line decoder module is shown in Figure 2. This module has as inputs the threshold detector outputs x_t and $\overline{x}_t$ (t=1,2,3) shown in Figure 1 and produces as outputs the one-out-of-4 selection values $f_0 = \overline{x}_1$, $f_1 = \overline{x_2 + x_1}$, $f_2 = \overline{x_3 + x_2}$, and $f_3 = x_3$. This two NOR gates module requires six devices for implementation in the form of 2 depletion mode and 4 enhancement mode devices. The total is then 42 devices per digit line, plus the 15 devices in the reference circuit overhead.
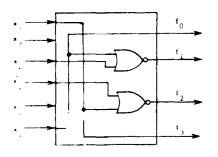
Fig. 2: One-out-of-4 decoder for LDTD outputs.

Table 2: Two digit line one-out-of-16 decoder.

$f_0 = \bar{x}_{11}\bar{x}_{01}$	$f_8 = \bar{x}_{01}\bar{x}_{13}x_{12}$
$f_1 = \bar{x}_{11}\bar{x}_{02}x_{01}$	$f_9 = \bar{x}_{02}x_{01}\bar{x}_{13}x_{12}$
$f_2 = \bar{x}_{11}\bar{x}_{03}x_{02}$	$f_{10} = \bar{x}_{03}x_{02}\bar{x}_{13}x_{12}$
$f_3 = \bar{x}_{11}x_{03}$	$f_{11} = x_{03}\bar{x}_{13}x_{12}$
$f_4 = x_{11}\bar{x}_{01}\bar{x}_{12}$	$f_{12} = \bar{x}_{01}x_{13}$
$f_5 = x_{11}\bar{x}_{02}x_{01}\bar{x}_{12}$	$f_{13} = \bar{x}_{02}x_{01}x_{13}$
$f_6 = x_{11}\bar{x}_{03}x_{02}\bar{x}_{12}$	$f_{14} = \bar{x}_{03}x_{02}x_{13}$
$f_7 = x_{11}x_{03}\bar{x}_{12}$	$f_{15} = x_{03}x_{13}$

DIGIT-LINES
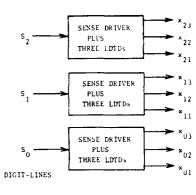
Fig. 3: Multiple digit line threshold detection.

An r=8 one-out-of-eight decoder using seven references V_{Rt} and seven LDTDs that provide the x_t's (t=1,2,···,7) can be designed by specifying the eight output functions $f_j(x_1,x_2,x_3,x_4,x_5,x_6,x_7)$, j=0,1,···,7, shown tabulated in Table 1. The device counts needed to implement the f_j's are 12 enhancement mode plus 6 depletion mode devices for a subtotal of 18. Counting the seven LDTDs, the totals are 98 devices per digit line plus constant overhead for the ROM of 35 devices to generate the references.

We now consider a sampling of multiple digit-line decoders at radix r=4, such as one-out-of-16 and one-out-of-32 decoders. The structure of such a digit line decoder requiring sensing of three digit lines S_0,S_1, and S_2 is shown in Figure 3. For clarity, the digit line sense driver is included in the module with the three LDTDs that produce the outputs x_{jt} (j=0,1,2 & t=1,2,3) and their complements, which are available but not shown. For a one-out-of-16 decoder we consider the S_1 and S_0 portion of Figure 3 and ignore S_2. For the one-out-of-32 decoder we consider two approaches. The first approach uses all of the S_0 and S_1 portions but only one third of the S_2 portion, which includes the sense driver and only one LDTD (comprising 3+11=14 devices) to provide the two values needed for binary switching. It might be possible to eliminate the LDTDs on S_2 altogether if two of the possible four ROM cell sense voltages are directly usable binary signals, but we shall be conservative and ignore this possibility. Two variations of this first approach are considered. The second approach uses all three sets of LDTDs on S_2, S_1, S_0, to generate the decoding in a manner that uniformly places a sense driver and three LDTDs (3+33=36 devices) at each digit line output.

Expressions for the r=4 one-out-of-16 two digit-line decoder are given in Table 2 for the sixteen functions in six variables of the form $f_j(x_{13},x_{12},x_{11},x_{03},x_{02},x_{01})$ for j=0,1,···,15. A total of 64 devices in the form of 16 depletion mode and 48 enhancement mode transistors are needed to implement the 16 NOR gates specified by the expressions in Table 2 for the one-out-of-16 decoder. The total device count for this decoder, excluding the 15 devices in the reference set, is then (3+33+3+33+64)=136 devices.

Expressions for a one-out-of-32 decoder for three digit lines with r=4 can be derived similarly. One has the option of using only one LDTD to generate x_{21}, but one is then restricted to the first 32 valid combinations (out of the 128 possible); this requires 232 devices to generate the $f_j(x_{21},x_{13},x_{12},x_{11},x_{03},x_{02},x_{01})$'s plus 14 devices in the LDTD for x_{21} only for a total of 246 devices. An alternative similar approach requiring an extra layer of decoding delay for binary switching uses x_{21} and its complement to control 64 enhancement mode devices on the f_i outputs of a one-out-of-16 decoder connected to the lower order digit lines S_1, S_0 so as to switch f_i between f_i and f_{i+16} based on the value of x_{21}. This scheme which employs a form of bit steering is shown in Figure 4, and requires a total of 214 devices for implementation, excluding the 15 devices in the reference set.

Fig. 4: symmetric one-out-of-32 decoder for three digit line 4-valued ROM fields.

The second approach to building a one-out-of-32 decoder at r=4 directly decodes the 64 (out of 512 possible) combinations of the nine x_{jt}'s (j=0,1,2 & t=1,2,3) to generate the 32 output functions desired which are of t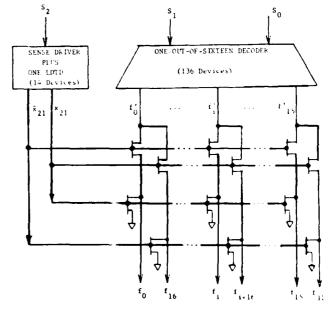he form $f_i(x_{23}, x_{22}, x_{21}, x_{13}, x_{12}, x_{11}, x_{03}, x_{02}, x_{01})$. We shall specify the desirable functions among these 64 in terms of the decimal representation of the binary row in the truth table on which each must take the value 1. We do this in terms of the number of inputs required on the NOR gate needed to generate the corresponding f_i. Eight choices lead to 3-input gates (namely, input combinations corresponding to rows 0, 7, 56, 63, 448, 455, 504, and 511). Twenty-four choices require 4-input gates for implementation (namely, those at rows 1, 3, 8, 15, 24, 31, 57, 59, 64, 71, 120, 127, 192, 199, 248, 255, 449, 451, 456, 463, 472, 479, 505, and 507). These choices provide a full one-out-of-32 decoding. We can choose fewer than twenty-four of these 4-input gate combinations to further reduce device counts if more than 16 but fewer than 32 decoder outputs are needed. A full 32 output decoding requires (8X4 + 24X5 + 3X36) = 260 devices excluding the 15 in reference set.

In the next section we consider how one might make use of these decoders using optimal and suboptimal groupings of microoperations at radix r=4 and at r=8 for the DEC PDP-11/40 based control ROM specification.

IV. An Example with Comparisons

A D=15 digit optimal radix r=4 grouping of microoperations suitable for encoding and storage in a 4-valued ROM is shown in Table 3. This 256 word, 75 microoperation control ROM specification for the MPP-11/40 is derived as a direct decoding of 43 of the 56 bits in the DEC PDP-11/40 control ROM specification as presented in [5]. An optimal radix 8 grouping of microoperations as well as the earlier suboptimal radix 4 grouping for this same example were presented in [5] but are reproduced in Table 3 for comparison. Over 15 and one half million nodes in the branch and bound search tree were examined (indicated by CLB = 15,522,470) before the algorithm declared this D=15 digit solution as optimal at r=4. The solution shown

was found at node 15,441,643, which illustrates the utility of the lower bound digit dimension predictions available in the parameterized table derived and presented in [7]-[10]. Had the branch and bound algorithm [11] used the lower bound prediction of $D_{\ell b}=15$ to terminate the search at node 15,441,643 when the optimal solution was found, it could have avoided searching an additional 80,827 nodes.

Our implementation of the branch and bound algorithm is still searching for an r=2 microoperation grouping pattern that improves upon the D=28 grouping found by Jeng [7] and presented in [5]; so we are forced to use this as the best known binary grouping for purposes of comparison.

Multiple-valued MOS ROM and decoder device counts for various encoding schemes are summarized in Table 4 for this MPP-11/40 example.

V. Conclusions

If speed of decoding is not a problem, then a two-bits per cell encoding of the best known D=28 bit binary microoperation grouping for the PDP-11/40 based example results in the lowest overall MOS transistor device count (excluding downstream binary decoders for the radix two encoded fields). A two-bits per cell encoding of the original 43-bit binary ROM specification, while reducing device count (by 38%) over the straight 43-bit binary implementation still does not achieve the savings available from suboptimal D=16 and optimal D=15 radix four direct encodings. Hence, the finding of optimal and reduced suboptimal direct encodings of microoperations can produce device count savings exceeding those available from a two-bits per cell approach. The new decoders presented in section III provide designers flexibility in the choice of either direct encoding or two-bits per cell encodings when reducing ROM word widths to save IC chip area. The techniques used to design them can be applied straightforwardly to radix three circuits and one-out-of-3^m decoders by reducing the number of LDTDs on each digit line by one. Implementation of reliable radix 8 (7, 6, & 5) ROM and decoder circuits, while appearing feasible in MOS technology, remain to be demonstrated.

Table 3: MPP-11/40 optimal and suboptimal microoperation groupings versus radix.

GROUP	r = 4; D = 15; Optimal CLB = 15,441,643 TOTAL NODES SEARCHED = 15,522,470	GROUP	r = 8; D = 12 CLB = 23,040 Optimal	GROUP	r = 4; D = 16 CLB = 38,896 Suboptimal
1	9 20 24 25 27 30 38 40 41 47 50 51 53 60 69	1	1 10 17 37 41 53 69	1	1 2 19 22 30 32 37 39 40 41 45 46 52 70 75
2	3 4 6	2	2 3 4 5 6 31	2	3 4 6
3	7 14 18 19 21 22 23 26 28 29 31 34 36 39 42 43 45 46 48 49 52 55 56 63 66 68 70 71 74 75	3	18 21 57 66 67 68 74	3	26 38 49 55 63 64 65 66 67 68 69 71 72 73 74
4	32 44 61	4	19 24 28 32 44 48 61	4	5 10 14 20 23 24 34 42 44 48 50 51 53 56 61
5	10 13 15	5	13 15 16 25 35 45 56	5	13 15 16
6	33 37 72	6	14 22 29 33 50 51 58	6	33 57 58
7	12 57 67	7	7 12 46 52 60 70 71	7	12 17 28
8	17 54 73	8	9 26 27 54 63 72 73	8	7 29 54
9	5 8 35	9	8 23 38 43 47 49 55	9	8 35 47
10	62 64 65	10	34 39 40 42 62 64 65	10	9 21 62
11	16 58 59	11	30 36 59 75	11	36 59 60
12	1 2 11	12	11 20	12	11 27 31
				13	18 25 43

253

Table 4: Device counts for control ROM encoding schemes.

Radix used for Grouping Microoperations	8	4	4	4	2	2	2	2
ROM cell and Encoding Radix r	8	4	4	4	2	2	4	4
Word Width D	12	15	15	16	28	43	14	23
ROM cell devices excluding select for rows and columns	3072	3840	3840	4096	7168	11008	3584	5888
Multiple-valued Decode Devices Excluding 2nd layer binary for 2-bits/cell	1211	831	785	843	--	--	561	887
Total Device Count	4283	4671	4625	4939	7168	11008	4145	6775
Notes	(1)	(2)	(3)	(4)	(5)	(6)	(5,7)	(6,7)

Notes:
(1) Optimal r=8 grouping and direct encoding with one-out-of-8 decoding.
(2) Optimal r=4 grouping and direct encoding with 10 one-out-of-4 decoders, 1 one-out-of-16 decoder, and 1 one-out-of-32 decoder (260 dev.).
(3) Same as (2) except using one 214 device one-out-of-32 decoder.
(4) Suboptimal r=4, D=16 grouping and direct encoding with 10 one-out-of-4 decoders and 3 one-out-of-16 decoders.
(5) Best known D=28 binary encoding from [5].
(6) PDP-11/40 actual implementation.
(7) Two-bits per cell encoding.

References
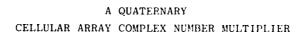
[1] C. Mead and L. Conway, Introduction to VLSI Systems, Reading, MA; Addison-Wesley, 1980.

[2] C. McMinn, "The Intel 8087: a numeric data processor," in 1980 Electro Professional Program Papers, Boston, MA, May 1980, pp. 14/5; 1-8.

[3] M. Stark, "Two bits per cell ROM," Digest of Papers 22nd IEEE Computer Society Intl. Conf., (COMPCON Spr '81), San Francisco, CA, Feb. 1981, pp. 209-212.

[4] J. Bayliss, J. Deetz, C. Ng, S. Ogilvie, C. Peterson, and D. Wilde, "The interface processor for the Intel VLSI 432 32-bit computer," IEEE J. of Solid-State Circuits, vol. SC-16, no. 5, Oct. 1981, pp. 522-530.

[5] C.B. Silio, Jr., J.H. Pugsley, B.A. Jeng, and L.H. Jones, "Further results on control store size reduction with multiple-valued encodings," in IEEE Proc. of 11th Intl. Symp. on Multiple-valued Logic, Oklahoma City, OK, May 1981, pp. 54-61.

[6] Digital Equipment Corp., PDP-11/40 Maintenance Manual, vol. II, Engineering Drawings, Maynard, MA, 1972, Drawing No. M7232-0-1, U Word, Sheets 9-12.

[7] B.R.A. Jeng, "Read only memory optimization for microprogrammed digital computers," PhD Dissertation, Univ. of Maryland, College Park, MD, 1979.

[8] B.A. Jeng and C.B. Silio, Jr., "Derivation of r-valued lower bound digit dimension for control store word width," in IEEE Proc. 12th Annual Southeastern Symposium on System Theory, Virginia Beach, VA, May 1980, pp. 128-132.

[9] C.B. Silio, Jr., J.H. Pugsley, and B.A. Jeng, "Control memory reduction using multivalued ROMs," in IEEE Proc. 9th Intl. Symp. on Multiple-valued Logic, Bath, England, May 1979, pp. 19-26.

[10] _____, "Control memory word width optimization using multiple-valued circuits," IEEE Trans. on Comput., vol. C-30, no. 2, Feb. 1981, pp. 148-153.

[11] J.L. Baer and B. Koyama, "On the minimization of the width of the control memory of micro-programmed processors," IEEE Trans. on Comput., vol. C-28, Apr. 1979, pp. 310-316.

[12] C.B. Silio, Jr., J.H. Pugsley, B.A. Jeng, and L.H. Jones, "Finding multiple-valued microoperation encodings to reduce control memory word width," in IEEE Proc. 10th Intl. Symp. on Multiple-valued Logic, Evanston, IL, June 1980, pp. 74-80.

[13] B.T. Murphy, "Cost-size optima of monolithic integrated circuits," Proc. of the IEEE, vol. 52, December 1964, pp. 1537-45.

[14] T.M. Whitney, "Microprocessor architecture," Chapt. 3 in Computer Architecture, H.S. Stone, ed., Chicago, IL: Science Research Associates, Inc., 1980, pp. 99-100.

[15] R. Nave and J. Palmer, "A numeric data processor," in 1980 IEEE Digest International Solid State Circuits Conf., Philadelphia, PA, Feb. 1980, pp. 108-109.

[16] W. Lattin, et al., "A methodology for VLSI chip design," LAMBDA, 2nd Quarter, 1981, pp. 34-44.

Note

The following transistor symbol ⊣ used in Figures 1 & 4 means ⊣ .

A QUATERNARY
CELLULAR ARRAY COMPLEX NUMBER MULTIPLIER

Tich T. Dao

Introduction.

Knuth (1) has proposed an unconventional single component representation of complex numbers whereby the radix is chosen to be purely imaginary $(2j)$ with the digit set comprised of the first four positive integers $(0,1,2,3,)$. This is in contrast with the conventional two components representation with binary radix whether in Cartesian or polar coordinates. As shown in a previous paper (2) this novel representation leads to simple and interesting arithmetic in the complex field.

In this paper we shall examine the design of a complex-number multiplier, organized in a regular cellular array. Such a structure is highly desirable for VLSI implementation.

Parallel Multiplier.

Without loss of generality assume that we are dealing with Gauss Integers which means complex numbers with real integer and imaginary integer only. An n digit in Knuth's representation of a complex number X is written as:

$$x = \sum_{k=-1}^{n-2} a_k (2j)^k$$

where all: $a_k = (0,1,2,3)$

except: $a_{-1} = (0,2)$

We can rewrite the above sum by grouping real elements separately from the imaginary elements; real ones coming only from the even power of 2j and the imaginary ones from the odd powers.

$$x = \sum_{l=0}^{(n-2)/2} a_{2l}(-4)^l + 2j \sum_{l=0}^{(n-2)/2} a_{2l-1}(-4)^{l-1}$$

In this form the real part of X is represented in base-4, as is its imaginary part. Since complex numbers are given and delivered in base 2, conversion back and forth between the two bases is necessary. Fortunately, it is simple to implement.

Before considering arithmetic at the word level, let us remind ourselves about arithmetic at the digits level, digit taking values in $(0,1,2,3,)$.

1. Digit Sum:

$$c_{k-1} + x_k + y_k = s_k + 4c_k = s_k - c_k(2j)^2 \qquad (3)$$

The sum of two digits plus a carry produces a sum and a negative carry $-c_k$ to the two digit positions $c_k = (0,1)$

Similarly $-c_{k-1} - x_k - y_k = -s_k + c_k(2j)^2 \qquad (4)$

2. Digit Difference:

If we define the 4's complements as:

$$\bar{y}_k = 4 - y_k \qquad (5)$$

then the difference is reduced to a sum of the minuend with the complement of the Subtrahend (6)

$$x_k - y_k = x_k + (4 - y_k) + (2j)^2$$

with an extra carry + 1

$$= x_k + \bar{y}_k + (2j)^2$$

3. Digit product:

Product of two digits $x_k \cdot y_l$ produces a sum $s_{k,l}$ together with a negative carry $c_{k,l}$ to the digit at two positions above:

$$x_k \cdot y_l = s_{k,l} - c_{k,l}(2j)^2$$
$$s_{k,l} = (0,1,2,3)$$
$$c_{k,l} = (0,1,2)$$

To arrive at a cellular array organization of a multiplier consider first the full parallel structure.

Given two words X and Y in quaterimaginary representation their product is defined as:

$$x \cdot y = \sum_{k=-1}^{n-2} x_k (2j)^k \cdot \sum_{l=-1}^{n-2} y_l (2j)^l \qquad (8)$$

$$= \sum_l [\sum_k x_k (2j)^k] y_l (2j)^l$$

In this above form the product is performed as in a pencil-paper fashion, one digit at a time of the multiplier with the partial product:

$$[\sum_k x_k (2j)^k] y_l \cdot (2j)^l = \sum_k x_k y_l (2j)^{k+l} \qquad (9)$$

the digit product $x_k \cdot y_l = s_{k,l} - c_{k,e} (2j)^2$

or: $x \cdot y = \sum_l [\sum_k [s_{k,l}(2j)^k - c_{k,l}(2j)^{k+2}]](2j)^l$

$$\qquad (10)$$

Assume for the sake of illustration a 4 x 4 multiplication. The digit arrangement is shown in Fig. 1. Each partial product generates two rows of digits: the upper row represents the sum digits and the lower row the carry digits (0,-1,-2), shifted two positions toward the MSB.

Digits of the same weight are lined up along the same column. They are added in the carry-save mode as in binary multiplication. Notice that digits are alternatively positive and negative, therefore it would be natural to separate them out into two groups adding positive digits producing a negative carry and adding negative digits producing a positive carry.

By referring to the Table 1, let us consider the first partial sum. To this we add the second partial product producing the second partial sum. As seen, all rows in between are the intermediate sums. We proceed the same way for the remaining partial sums. At the last partial sum, we have to reduce the two rows of opposite signs which represent the result into one single row of positive digits. To implement the above multiplication, we need for each iteration (partial product followed by partial sum) a row of arithmetic cells comprised each of a digit product and a pair of two bits full adders, one for the positive digits, the other for the negative digits. (Fig 1)

Negative carry from positive adder is fed to the corresponding negative adder and the positive carry from the negative adder is fed to the corresponding adder. In the schematic, thin line drawing relates to positive arithmetic and the heavy line drawing to the negative arithmetic. When all positive digits result is required then the last iteration will terminate in a subtraction stage. The arrangement shown in Fig. 1 is akin to an array of cells with interleaved connections.

Cellular Array Multiplier - Accumulator.

A more regular array for a complex multiplier plus accumulator can be derived by generalizing the Guild cell for binary multiplication.

In binary arithmetic that cell[2] is defined by the following at the bit level:

$$x+y+a+b = (c2+b)$$

where x,y,a,b are binary inputs

and c,s are binary outputs.

In quaterimaginary arithmetic, we would have:

$$x+y+a+b = c(2j)^2 + s$$

where x,y,a,b are quaternary inputs and c,s are quaternary outputs.

a,b,c,s can also take negative values.

We define by a^+, a^- respectively the positive and negative value of a.

By definition $x \cdot y = c_{xy}^-(2j)^2 + s_{xy}^+$

We can rewrite the above equation as:

$$c_{xy}^-(2j)^2 + s_{xy}^+ + a + b = c(2j)^2 + s$$

For ease of implementation we split the inputs into positive and negative groups, then the functional equations of the cell will be:

$$s_{xy}^+ + a^+ + b^+ = c^-(2j)^2 + s^+$$

$$s_{xy}^- + a^- + b^- = c^+(2j)^2 + s^-$$

We can realize the complex arithmetic cell with two separated cells defined respectively by the above equations. Since c_{xy} generated by the same inputs x, y which produce s_{xy}^+ has a $(2j)^2$ weight higher, it can only be added to a^- and b^- of the corresponding weight.

Consider Fig. 2 which shows an array built around the two cells performing the operation X. Y + Z. Again the light drawing refers to positive digits operands; heavy drawing refers to negative digits operands. Each cell is labelled by two digits $x_i.y_j$. Depending on the polarity of the cell, the arithmetic operation performed is either:

$$4c+s = (x_i.y_j) \bmod 4 + A + B$$

$$4c+s = x_i.y_j - (x_i.y_j) \bmod 4 + A \pm B$$

We also need at the edge of the array few adder cells with 3 to 4 inputs.

Since positive and negative digits are added separately the intermediate result is given by two sequences of positive and negative digits which are again added together to provide the final result as a sequence of positive digits only.

Each adder is defined by:

$$a - b + c_i = s + c_o (2.j)^2$$

with all $a,b,s = (0,1,2,3)$

$$c_i, c_o = 0, \pm 1$$

To assume that S is positive in the case whereby:

$$a - b + c_i < o$$

we replace the S and C_o by their 4's complements.

Fig. 3 illustrates a numberical example of the product of two numbers:

$(-11-4j) \times (-6-22j) = -22 + 266j$

$<1321> . <3212> = <1021133212>$

Also Fig. 4 shows that one can add to this product one number, for example $(-11-4j)$ such that:

$(-11-4j).(-6-22j)+(-11-4j) = (-33+262j)$

or:

$<1321>.<3212>+<1321> = <102102133>$

The array actually performs the operation: x.y+A+B or one product followed by two additions. The dot lines are extra carry connections between cells when such an operation is desired.

Let us compare the standard X.Y parallel multiplier with the X.Y + A+B. In Fig. 1, at the left upper hand corner, we have the building block which is a digit product associated with a positive digit adder and a negative digit adder. If we incorporate the sum logic part of the digit product into the positive adder and the carry logic into the negative adder, would have two arithmetic cells similar to those in Fig. 2.

The number of those cells in the X. Y multiplier is 26, while it is 32 plus 2 extra straight adders in the X.Y + A + B. However, to perform the same operation as in the latter case, an extra row of 10 digits ripple adder is required.

Then the two approaches have the same complexity: 43 adders versus 45 adders. The difference resides in the propagation delay. The Guild cell array performs the product X . Y and the accumulation of A and B simultanously while in the standard case accumulation of A and B is only done when the product has been completed.

A similar binary parallel complex-number multiplier and accumulator would require 4 copies of a 4 x 4 parallel multiplier plus two copies of an 8 bits 3 operand adder.

Since a 4 x 4 multiplier uses about 16 full adders and the 3 operand adder uses also 16 full adders, the total number of full adders required would be 96 or equivalently 48 quaternary full adders.

Therefore, the difference in hardware complexity between binary and quaternary approaches appears to be not significant. This would be true if quaternary full adder is implemented with binary components. When the same adder is implemented with multivalued logic as described in previous papers (2,4), the picture looks quite different. Since the ratio of component complexities between a quaternary full adder designed with binary elements and that designed with multivalued is greater than 2, therefore Knuth's quaterimaginary cellular array approach to the complex number product and accumulation would be highly recommended.

Conclusion:
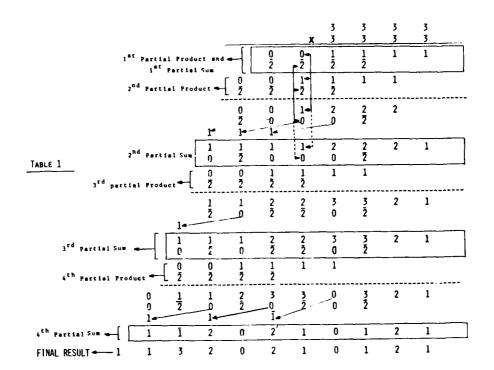
We have reviewed briefly the Knuth's quaterimaginary representation of a complex-numbers and proposed two alternate implementations of a parallel multiplier and accumulator, commonly used in signal processing and in particular in F.F.T. The cellular array approach proved to be more efficient and much denser than the binary equivalent implementation when multivalued logic components are used.
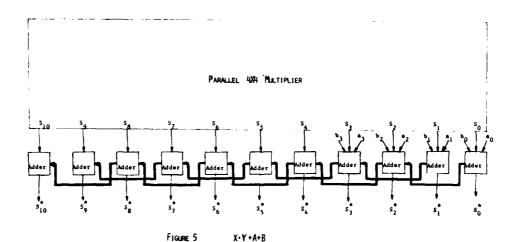
References:

- 1.   D.E. Knuth,: "An Imaginary Number System", Com. Ass. Comp. Mach., Vol. 3, pp. 245-247, 1960.

- 2.   T.T. Dao,: "Knuth's Complex Arithmetic with Quaternary Hardware", Proc. of the 12th ISMVL, pp. 94-98, 1982.

- 3.   H.H. Guild,: "Fully Iterative Fast Array for Binary Multiplication and

Addition", Elect. Letters, Vol. 5,
pp. 263, June 1969.

- 4. T.T. Dao @ Al., "Complex Number
     Arithmetic with Odd-Valued Logic",
     I.E.E.E. Trans. on Computers, Vol.
     C. 29, N° 7, pp.604-610, July 1980.

TABLE 1

							X	3 3	3 3	3 3	3 3

1st Partial Product and 1st Partial Sum

$$\frac{0}{2} \quad \frac{0}{2} \quad \frac{1}{2} \quad \frac{1}{2} \quad 1 \quad 1$$

2nd Partial Product

$$\frac{0}{2} \quad \frac{0}{2} \quad \frac{1}{2} \quad \frac{1}{2} \quad 1 \quad 1$$

$$\frac{0}{2} \quad \frac{0}{0} \quad \frac{1}{0} \quad \frac{0}{2} \quad 2 \quad 2$$

2nd Partial Sum

$$1 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2 \quad 1$$
$$0 \quad \frac{1}{2} \quad 0 \quad 0 \quad 0 \quad \frac{2}{2}$$

3rd Partial Product

$$\frac{0}{2} \quad \frac{0}{2} \quad \frac{1}{2} \quad \frac{1}{2} \quad 1 \quad 1$$

$$\frac{1}{2} \quad 0 \quad \frac{2}{2} \quad \frac{2}{2} \quad 0 \quad \frac{3}{2} \quad 2 \quad 1$$

3rd Partial Sum

$$1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 2 \quad 1$$
$$0 \quad \frac{1}{2} \quad 0 \quad \frac{2}{2} \quad \frac{2}{2} \quad 0 \quad \frac{3}{2}$$

4th Partial Product

$$\frac{0}{2} \quad \frac{0}{2} \quad \frac{1}{2} \quad \frac{1}{2} \quad 1 \quad 1$$

$$0 \quad \frac{1}{2} \quad 1 \quad 2 \quad 3 \quad 3 \quad 0 \quad \frac{3}{2} \quad 2 \quad 1$$
$$0 \quad 0 \quad 0 \quad 2 \quad 0 \quad 2 \quad 0$$
$$1 \quad 1 \quad \bar{1}$$

4th Partial Sum

$$1 \quad \bar{1} \quad 2 \quad 0 \quad 2 \quad 1 \quad 0 \quad 1 \quad 2 \quad 1$$

FINAL RESULT

$$1 \quad 1 \quad 3 \quad 2 \quad 0 \quad 2 \quad 1 \quad 0 \quad 1 \quad 2 \quad 1$$

PARALLEL 4X4 MULTIPLIER

S_{10} S_9 S_8 S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0

Adder Adder Adder Adder Adder Adder Adder Adder Adder Adder Adder

$b_3 a_3$ $b_2 a_2$ $b_1 a_1$ $b_0 a_0$

s_{10}^* s_9^* s_8^* s_7^* s_6^* s_5^* s_4^* s_3^* s_2^* s_1^* s_0^*

FIGURE 5   X·Y+A+B

FIGURE 1
PARALLEL 4X4 MULTIPLIER

259

ARRAY X·Y+Z

Figure 2

260

ARRAY Y.Y+Z   Figure 3

261

Figure 4   ARRAY Y.Y+Z

262

Session 6B
Fuzzy Logic

A COMPARISON OF FUZZY SWITCHING FUNCTIONS AND

MULTIPLE - VALUED SWITCHING FUNCTIONS

David R. Luginbuhl and Abraham Kandel

Department of Mathematics and Computer Science
The Florida State University
Tallahassee, Florida 32306

ABSTRACT

The fact that a fuzzy switching function may take on infinitely many values is a good reason to examine the relationship between fuzzy switching functions and multi-valued (or multiple-valued or many-valued) switching functions. There are many aspects of the two types of functions that can be compared.

After presenting a brief introduction to both multi-valued switching functions and fuzzy switching functions, we will compare some of the features of the two types of functions. We will then propose an algorithm which minimizes fuzzy functions, using techniques previously applied only to multi-valued switching functions.

1. INTRODUCTION

Although there has been much work done in the field of fuzzy logic (1) and multi-valued logic (7) to cover the inadequacies of binary logic, little has been done to show the relationship between the two. This is surprising, considering the similarities between fuzzy switching functions and multivalued logic switching functions.

The purpose of this paper is to establish a relationship between the two types of switching functions. Following are general descriptions of fuzzy switching functions and multi-valued switching functions. The next chapter focuses on a comparison of the properties of the two types of functions. Finally, an algorithm for the minimization of fuzzy functions is presented. This algorithm is based on an algorithm used to minimize multi-valued switching functions.
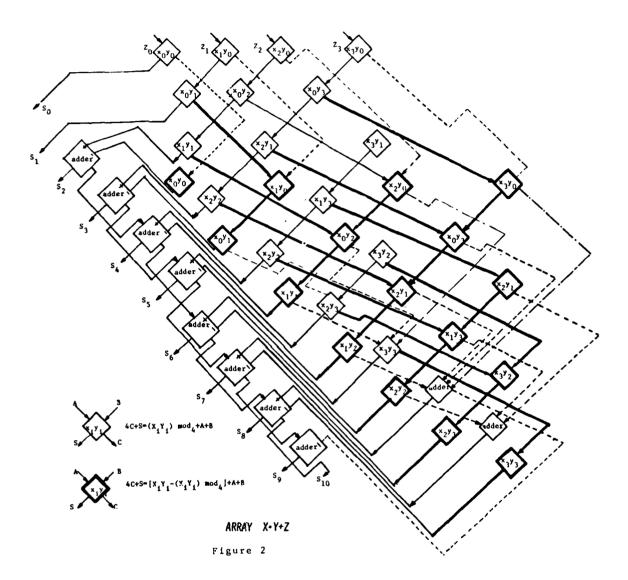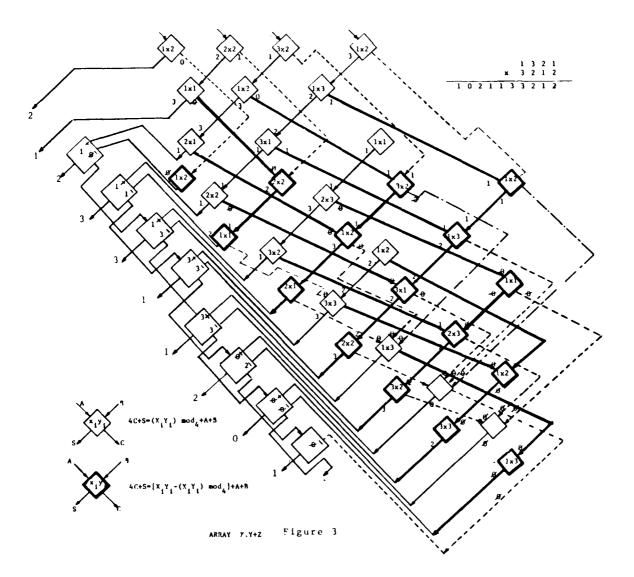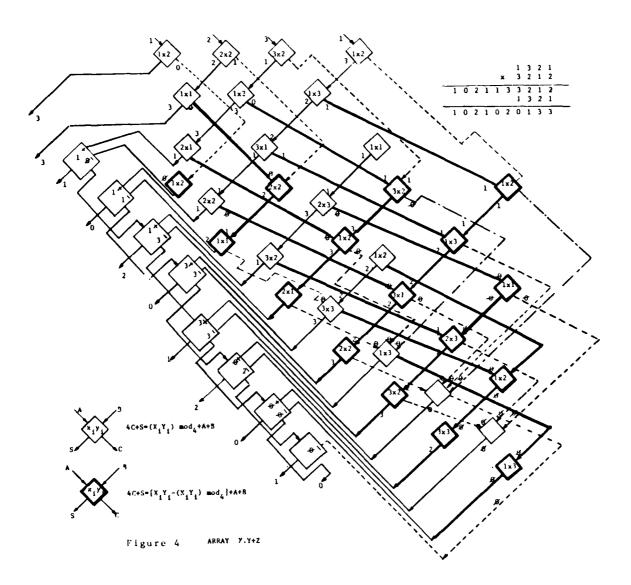
The concept of fuzzy sets, proposed by Zadeh (9), has led to work with fuzzy switching functions. The definition of a fuzzy set follows. Let $X = (x)$ be a space of objects. A fuzzy set A in X is a set of ordered pairs $A = ((x, \mu_a(x)))$, x in X, where $\mu_a(x)$ is the grade of membership of x in A. For simplicity, we assume that $\mu_a(x)$ is a real number in the interval $[0,1]$, with 1 representing membership and 0 representing non-membership in a fuzzy set. In this paper, we will use the term "fuzzy variable" for the term "membership grade" of a variable in a set; that is, x will represent $\mu_a(x)$ (4).

The definition of a fuzzy algebra can be found in several papers (4,8). It is a distributive lattice with existence of unique identities under + and * (3). Unlike a Boolean algebra, there does not exist a complement x' for every x such that $x*x' = 0$ and $x + x' = 1$. All that is known is that $x*x' \leq \frac{1}{2}$ and $x + x' \geq \frac{1}{2}$.

The particular fuzzy algebra we will use will be defined by the system $([0,1],+,*,')$, where $+,*,$ and $'$ are interpreted as max, min, and complement $(x' = 1 - x,$ for x in $[0,1])$, respectively. We will use the convention of writing $x*y$ as xy.

As in (4), a fuzzy switching function (FSF) will be defined as a function from V^n to V, where $V = [0,1]$, represented as a logic formula constructed from the logic operations, *, +, and ', as described above, applied to fuzzy variables $x_1,\ldots,x_n$, and the constants 0 and 1. Fuzzy forms, generated by the n fuzzy variables, are defined recursively as follows:

a) 0 and 1 are fuzzy forms.
b) A fuzzy variable x_i is a fuzzy form.
c) If A is a fuzzy form, then so is A'.
d) If A and B are fuzzy forms, then so are A+B and AB.
e) Nothing else is a fuzzy form.

We will represent FSF's using this definition of fuzzy forms (4).

The multi-valued logics (also known as multiple valued logics or many-valued logics) discussed in this paper are the standard sequence S_n, which is defined in Rescher (7), and the Allen-Givone algebra defined in (1). The truth rules of S_n are as follows:

$$/p'/ = 1 - /p/$$

$$/p \wedge q/ = min(/p/,/q/)$$

$$/p \vee q/ = max(/p/,/q/).$$

For the purposes of this discussion, implication and equivalence are omitted, since these are not defined as truth rules in the fuzzy algebra.

In the Allen-Givone (A-G) switching algebra, the variables assume one of m values, designated as $v_1, v_2, \ldots, v_m$, where $v_1 < v_2 < \ldots < v_m$, so that

we can assign 0 to v_1, 1 to v_2, ..., m-1 to v_m. Therefore, input and output variables take on values from $(0,1,...,m-1)$. The operations or truth rules are the same as those for the fuzzy algebra:

$$x+y=\max (x,y) \qquad x \cdot y=\min (x,y),$$

where x and y are elements of $(0,1,...,m-1)$. Several properties of the A-G algebra, such as idempotency and commutativity, are listed in (1).

Of the possible unary operators in the A-G algebra, the operator we will use is the A-G literal $X(a,b)$, defined as follows:

$X(a,b) = 0$ if the value of $X < a$ or the value of $X > b$, m-1 if a $\leq$ the value of $X \leq b$,

where a and b are in $(0,1,...,m-1)$ and a $\leq$ b.

The literal described above is used to form A-G product terms. These product terms are denoted by

$r \cdot X_{i1}(a_{i1},b_{i1}) \cdot X_{i2}(a_{i2},b_{i2}) \cdots X_{ik}(a_{ik},b_{ik})$; that is,

the minimum of a constant r (r in $(0,1,...,m-1)$) and a set of literals, where a variable X_i appears at most once. An example in a 4-valued system is $2 \cdot X_1(0,1) \cdot X_2(1,3)$. This would have the value 2 when $X_1 = 0$ or 1 and $X_2 = 1$, 2, or 3. It would have the value 0 otherwise.

For the purposes of minimization, literals of the form $X_i(0,m-1)$ can be eliminated. This is because literals of this form equal m-1 for all values of X_i, and therefore have no effect on the product term. For the map minimization associated with the Allen-Givone algebra, we will make use of the Generalized Logic Diagram (GLD), described by Michalski (6).

2. COMPARISON OF PROPERTIES

Conceptually, there is a difference between fuzzy logic and multi-valued logic. The basic concern of the latter is the amount of truth or falsity of statements, although this has been expanded in applications. Fuzzy logic is concerned with the grade of membership of items in a given set.

This difference can be resolved, however, if we "alter" the definition of fuzzy logic slightly. We could say that fuzzy logic deals with the amount of truth or falsity of a statement that an item is in a given set. Although this might be considered "cheating" by some, it does give us a way to think of fuzzy logic and multi-valued logic as being more closely related.

To take this a step further, we would like to think of multi-valued switching functions (MVSF's) as FSF's. One problem with this is that MVSF's generally take on discrete values, while fuzzy values are continuous between 0 and 1. However, if we consider the standard sequence of many-valued generalizations of Łukasiewicz (7), we will find special systems $S_{\aleph}$ and $L_{\aleph}$, which are in fact infinitely valued systems. Therefore, we see that there can be MVSF's which, like FSF's, take on continuous values.

Another way to handle the problem that FSF's pose because of their use of continuous values is

proposed by Marinos (5). He introduces a way to give FSF's a "decision mechanism", so that the value of the function can be used, as in boolean logic or MVL, to make a decision about a particular event (he notes, for example, the fact that a FSF has a value of, say, 0.3 does not mean much unless there is some outcome dependent on that fact). The concept he proposes is a fuzzy function classification. The continuous range of fuzzy values is subdivided into a finite number of classes as follows:

Class 1: $a_1 < x < 1$

Class 2: $a_2 \leq x \leq a_1$

.

.

.

Class n: $0 < x < a_{n-1}$,

where $1 > a_1 > a_2 ... a_{n-1} > 0$. A fuzzy value or function can now be assigned to one of these classes, depending on which value it assumes in the region between 0 and 1. Through the use of this class system, FSF's can utilize properties of ordinary logic (including multi-valued logic). The remainder of Marinos' paper shows how this classification approach can be used in the analysis and synthesis of FSF's, including the use of MVL in creating fuzzy logic circuits. With Marinos' classification system, FSF's can easily take on the appearance of MVSF's.

However, another problem arises when considering the truth rules of fuzzy algebras and multi-valued algebras. It has already been shown that conjunction and disjunction are the same in fuzzy algebra and in the standard sequence. The property of negation is also the same. There are, however, no truth rules for implication or equivalence in the basic definition of FSF's. For the purpose of this discussion, this problem is not major, since we are not really concerned with implication or equivalence in this paper. Even if these rules were pertinent to this paper, we could simply define fuzzy rules for implication and equivalence. There is an advantage to this, since we could pattern these rules after the rules of the particular multi-valued system we were examining.

It is interesting to see if and how fuzzy algebra could be used in conjunction with the Allen-Givone (A-G) switching algebra. There are many similarities between the two, such as the definitions for conjunction and disjunction. Also, many algebraic properties, such as idempotency, commutativity, absorption, etc., are the same in both systems. There are, of course, differences. For example, the logical values in the switching algebra correspond to the whole numbers up to one less than however many values are being used. The fuzzy values are all between 0 and 1, inclusive.

This difference can be resolved by letting A-G values correspond to values in the standard sequence. This can be accomplished with virtually no effect on the switching algebra, since there are

still the same number of values in the same order.

The major difference is, once again, the fact that the A-G algebra was designed with a finite number of truth values in mind. It is difficult to use the unary operator X(a,b) to represent a FSF, because of the infinite number of values in FSF's. However, a way to overcome this problem, and in fact represent FSF's using the unary operator, is demonstrated in the next chapter.

3. MINIMIZATION OF FSF'S USING MVL TECHNIQUES

Because of the relation of FSF's to MVSF's, a minimization technique applied to MVSF's should be effective on FSF's. The techniques used by Allen and Givone (1) or Michalski (6) can in fact be adapted to fuzzy functions. Later in this chapter, a minimization procedure using these techniques is described. Before it is given, several points of translation between fuzzy algebra and the A-G algebra must be made.

The procedure is based on the result of Thum (8) that there is a one-to-one correspondence between FSF's and a subset of functions

$f: (0,\frac{1}{2},1)^n \rightarrow (0,\frac{1}{2},1)$ that fulfill certain properties. This result means that we only have to consider ternary logic functions in the procedure.

With the sufficiency of ternary logic established, several propositions concerning the A-G algebra must be proven. These propositions deal with product terms in minimized form, and especially with the use of the unary operator

$X(a,b)$ ($^a X^b$ in (6)), which Allen and Givone call a literal.

Proposition 1

A-G literals in product terms with constant 1 must be of the form $X_i(a_i,a_i)$, where $a_i = 0$ or 1.

Proof: The combinations of intervals available in ternary logic are (0,0), (0,$\frac{1}{2}$), ($\frac{1}{2}$,$\frac{1}{2}$), ($\frac{1}{2}$,1) and (0,1). The interval (0,1) will not be used, because if a literal can take on every value available, it can be eliminated from the product. Except for (0,0) and (1,1), all intervals include $\frac{1}{2}$. However, if a literal assumes the value $\frac{1}{2}$, the value of the product term containing the literal can be no more than $\frac{1}{2}$. Therefore, the only intervals available for product terms with constant 1 are (0,0) and (1,1).

Proposition 2

An A-G product term containing a literal of the form $X_i(\frac{1}{2},\frac{1}{2})$ cannot contain literals of the form $X_j(0,0)$ or $X_j(1,1)$.

Proof: Assume that a product term containing $X_i(\frac{1}{2},\frac{1}{2})$ also contains $X_j(0,0)$ ($X_j(1,1)$). Since X_i assumes only the value $\frac{1}{2}$, the product term must have the value $\frac{1}{2}$. If X_j had the value $\frac{1}{2}$, the value of the product term would not be affected. The interval of X_j could be written as

$X_j(0,\frac{1}{2})$ ($X_j(\frac{1}{2},1)$), which is minimized further than

$X_j(0,0)$ ($X_j(1,1)$).

Proposition 3

If a literal of the form $X_i(0,\frac{1}{2})$ ($X_i(\frac{1}{2},1)$) occurs in a product term which contains no literal of the form $X_j(\frac{1}{2},\frac{1}{2})$, then in the same function, there is a literal of the form $X_i(0,0)$ ($X_i(1,1)$) in a product term with constant 1.

Proof: If a product term contains no $X_i(\frac{1}{2},\frac{1}{2})$ and contains $X_i(0,\frac{1}{2})$ ($X_i(\frac{1}{2},1)$) and has constant $\frac{1}{2}$, then there is input consisting of 0's and 1's, where $x_i = 0$ ($x_i = 1$), which gives the function a value of at least $\frac{1}{2}$. However, one of the properties of FSF's shown in (8) is that a function takes on the value 0 or 1 with input from $(0,1)^n$. Thus, with input from $(0,1)^n$, where $x_i = 0$ ($x_i = 1$), the function must assume the value 1. This means that there must be a product term with constant 1 containing the literal $X_i(0,0)$ ($X_i(1,1)$).

Proposition 4

(a) In a product term with constant 1, and A-G literal $X_j(1,1)$ corresponds to a fuzzy literal x_j.
(b) In a product term with constant 1, an A-G literal $X_j(0,0)$ corresponds to a fuzzy literal x_j'.

Proof: (a) If $X_j(1,1)$ occurs in a product term with constant 1, then the product term is 1 whenever $x_j = 1$. For a product term to equal 1, all of its literals must equal 1. Therefore, x_j must be one of the literals in the product term.

(b) If $X_j(0,0)$ occurs in a product term with constant 1, then the product term is 1 whenever $x_j = 0$. For a product term to equal 1, all of its literals must equal 1. If $x_j = 0$, then $x_j' = 1$, so x_j' must be one of the literals in the product term.

Proposition 5

In a product term, an A-G literal $X_i(\frac{1}{2},\frac{1}{2})$ corresponds to a fuzzy phrase $x_i x_i'$.

Proof: If a product term contains $X_i(\frac{1}{2},\frac{1}{2})$, that product term must be 0 for $x_i = 0$ or 1, otherwise the interval for x_i would include 1 or 0. The only way for the term to equal $\frac{1}{2}$ for $x_i = \frac{1}{2}$, and at the same time equal 0 for $x_i = 1$ or 0, is for $x_i x_i'$ to be included in the product term.

Proposition 6

In product terms containing A-G literals of the form $X_i(\frac{1}{2},\frac{1}{2})$:

(a) an A-G literal $X_j(\frac{1}{2},1)$ corresponds to a fuzzy literal x_j.
(b) an A-G literal $X_j(0,\frac{1}{2})$ corresponds to a fuzzy literal x_j'.

Proof: (a) Because the product term contains $X_j(\frac{1}{2},1)$, it must take a value of at least $\frac{1}{2}$ when $x_j = 1$. There are only three forms that x_j can take in a fuzzy product term: it can appear uncomplemented (as x_j), complemented (as x_j'), or as the minimum of x_j and x_j' ($x_j x_j'$). The last two forms could not occur in the product term since both contain x_j': and when $x_j = 1$, $x_j' = 0$, which means the product term would be 0. Therefore, x_j must occur alone in the product term.

(b) Because the product term contains $X_j(0,\frac{1}{2})$, it must take a value of at least $\frac{1}{2}$ when $x_j = 0$. Looking again at the three ways that x_j can occur in a fuzzy product term, x_j alone and

$x_j x_j'$ must be ruled out since both contain x_j, and when $x_j = 0$, the product term would be 0. Therefore, x_j' must occur alone in the product term.

With these propositions in mind, the algorithm for minimizing FSF's using MVL techniques can now be given.

Algorithm:

Part A: The values of the FSF must be determined for all input from $(0,\frac{1}{2},1)^n$. This can be done in several ways. Perhaps the best way is as follows:

(1) For each product term, determine what input is necessary to give the product term a value of 1. This means that each literal must have value 1. If the product term has value 1, the function will, too. Thus, give each of these conbinations value 1 (a map (1,6) can be used to keep track of the values).

(2) After (1) has been done for each product term, determine for each term what input would give value 0. For all other combinations (which are not already marked as 1), the value is $\frac{1}{2}$.

(3) After (2) has been done for each product term, any remaining input combinations have value 0. Once values have been determined for each input combination, the function can be written in the form used by Allen and Givone (1).

Part B: Minimize the function using a technique discussed in (1) (or (6)).

Part C:
(1) For all product terms with constant 1, convert all $X_j(a_j, a_j)$ to:

$$x_j, \text{ where } a_j = 1$$

$$x_j', \text{ where } a_j = 0.$$

(2) For all product terms which contain one or more literals of the form $X_i(\frac{1}{2},\frac{1}{2})$, convert all:

$$X_i(\frac{1}{2},\frac{1}{2}) \text{ to } x_i x_i'$$

$$X_j(0,\frac{1}{2}) \text{ to } x_j'$$

$$X_k(\frac{1}{2},1) \text{ to } x_k.$$

(3) Disregard remaining terms.

Proposition 7

The above algorithm will produce a minimized FSF.

Proof: Part A, which represents a FSF in a ternary form, has been shown to be correct on the basis of Thum's result (8). The result obtained from Part B is a minimal function in A-G form. For Part C, first consider all product terms with constant 1. These terms form the minimal way for the function to equal 1. So all must be accounted for. Proposition 1 limits the form of A-G literals in terms with constant 1 to $X_i(a_i, a_i)$, where $a_i = 0$ or 1. Proposition 4 shows how these are converted back to fuzzy literals (Algorithm, Part C(1)).
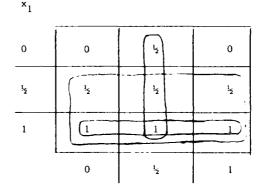
We next consider product terms with constant $\frac{1}{2}$; these must also be accounted for, unless they are accounted for by terms with constant 1. Proposition 3 shows that terms with constant $\frac{1}{2}$ which do not contain literals of the form $X_i(\frac{1}{2},\frac{1}{2})$ are covered by

corresponding terms with constant 1. Therefore, we only consider those terms with literals $X_i(\frac{1}{2},\frac{1}{2})$ (Algorithm, Part C(3)). Proposition 2 proves that only literals of the form $X_i(0,\frac{1}{2})$, $X_j(\frac{1}{2},1)$, and $X_k(\frac{1}{2},\frac{1}{2})$ can occur in such product terms. Propositions 5 and 6 show how these literals can be translated into fuzzy literals (Algorithm, Part C(2)). Since a minimized A-G function contains only product terms with constants 1 and $\frac{1}{2}$, all terms have been translated back to fuzzy terms, so the algorithm has produced a minimized FSF.

To show how the algorithm works, several examples will be presented:

Example 1:
$$F_1(x_1, x_2) = x_1 + x_1' x_2 x_2'$$

First, we set up a table of values (we will use the Generalized Logic Diagram (GLD) described in (6)) for x_1, x_2 in $(0,\frac{1}{2},1)$ (see figure 1). Note that the first product term (x_1) is 1 whenever $x_1 = 1$, and the second product term $(x_1' x_2 x_2')$ can never be 1 because it contains $x_2 x_2'$, which is less than or equal to $\frac{1}{2}$. So we mark all combinations where $x_1 = 1$ with a 1.

Next, note that the first product term is 0 whenever

x_1

$$x_1 + x_1' x_2 x_2'$$

Figure 1

$x_1 = 0$. We therefore mark all other combinations (not already marked) as $\frac{1}{2}$. The second term is 0 when $x_1 = 1$ or $x_2 = 0$ or 1. Mark all other combinations (not already marked) as $\frac{1}{2}$. The rest are marked 0.

Using the minimization technique discussed in (1), the function can be minimized to $F_1 = 1 \cdot X_1(1,1) + \frac{1}{2} \cdot X_1(\frac{1}{2},1) + \frac{1}{2} \cdot X_2(\frac{1}{2},\frac{1}{2})$. The first product term is transformed into x_1. This is because the function has the value 1 whenever x_1 is 1, and this means that x_1 must appear as a term with no other literal affecting it with a smaller value. The second term can be discarded, since a map of the term x_1 alone would show that the function is at least $\frac{1}{2}$ whenever x_1 is either $\frac{1}{2}$ or 1, which is what the second term implies. The last

term is written as x_2x_2', as directed by Part C(2) of the algorithm. The function can now be written as $F_1 = x_1 + x_2x_2'$, which is minimized FSF equal to the original function.

Example 2:

$$F_2(x_1,x_2,x_3) = x_1x_2 + x_1x_1'x_2'x_3 + x_1'x_2x_3$$

To set up the GLD for F_2 (figure 2), we first determine where the function is equal to 1. The first product term is 1 when both x_1 and x_2 are 1 (it does not matter what x_3 is equal to for this product term). The second term cannot be equal to 1 since it contains x_1x_1'. The third term is equal to 1 when x_1 is 0, and x_2 and x_3 are both 1. To determine where $F_2 = \frac{1}{2}$, we first determine where each product term is 0. The first term is equal to 0 when $x_1 = 0$ or $x_2 = 0$. We mark all other locations on the GLD (not already marked) as $\frac{1}{2}$. The second term is 0 when $x_1 = 0$ or 1, or when $x_2 = 1$, or when $x_3 = 0$. The last term is 0 when $x_1 = 1$, $x_2 = 0$, or $x_3 = 0$. After marking the appropriate locations $\frac{1}{2}$, we mark everything else as 0.

After minimization, we determine that $F_2 = 1 \cdot X_1(1,1)X_2(1,1) + 1 \cdot X_1(0,0)X_2(1,1)X_3(1,1) + \frac{1}{2} \cdot X_1(\frac{1}{2},1)X_2(\frac{1}{2},1) + \frac{1}{2} \cdot X_2(\frac{1}{2},1)X_3(\frac{1}{2},1) + \frac{1}{2} \cdot X_1(\frac{1}{2},\frac{1}{2})X_3(\frac{1}{2},1)$. The first two terms translate in a FSF to $x_1x_2 + x_1'x_2x_3$, as explained in Part C(1) of the algorithm. The third term is covered by x_1x_2 and can be discarded. Likewise, the fourth term is covered by $x_1'x_2x_3$ or x_1x_2, since one of these terms takes on a value of at least $\frac{1}{2}$ whenever both x_2 and x_3 take on values of $\frac{1}{2}$ or 1. The last term becomes $x_1x_1'x_3$, by Part C(2) of the algorithm. Therefore, $F_2 = x_1x_2 + x_1'x_2x_3 + x_1x_1'x_3$, which is the minimized form of F_2.

Example 3a:

$$F_3(x_1,x_2,x_3,x_4) = x_1x_2x_2'x_3x_4' + x_1'x_2x_2'x_3x_4'$$
$$+ x_1'x_2x_2'x_3'x_4 + x_1'x_2x_2'x_3'x_4 + x_1x_2x_2'x_3x_4$$
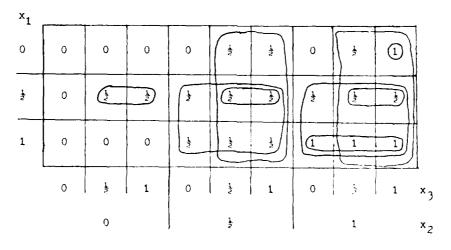
This FSF cannot take on the value 1, because x_2x_2' is common to all five terms. Therefore, to set up the GLD for this function (figure 3), we determine where it equals $\frac{1}{2}$ by determining where it does not equal 0. The first term does equal 0 when x_1, x_2, or x_3 equals 0, or when x_2 or x_4 equal 1. So for all other values of these literals, the value of the product term is $\frac{1}{2}$. This process is continued for each term.

Minimizing the function by this diagram, we have $F_3 = \frac{1}{2} \cdot X_1(0,\frac{1}{2})X_2(\frac{1}{2},\frac{1}{2})X_3(0,\frac{1}{2}) + \frac{1}{2} \cdot X_1(\frac{1}{2},1) X_2(\frac{1}{2},\frac{1}{2})X_3(\frac{1}{2},1) + \frac{1}{2} \cdot X_2(\frac{1}{2},\frac{1}{2})X_3(\frac{1}{2},1)X_4(0,\frac{1}{2})$. Part C(1) and (3) of the algorithm do not apply here. Part C(2) translates the function to

$$F_3 = x_1'x_2x_2'x_3' + x_1x_2x_2'x_3 + x_2x_2'x_3x_4', \text{ the}$$

minimized form of the function.

Example 3b:

For comparison purposes, we will minimize F_3 using the technique described in (4) (figure 4). This algorithm is based on the use of the fuzzy iterative consensus, discussed in (4). We convert each fuzzy literal to its decimal equivalent, after determining that no phrase of the function subsumes any other phrase. Table K is constructed with one row for each phrase of F_3. The first row of table K_1 is created from the first two rows of table K. Since the phrase represented by this row is subsumed by the first two phrases, they are removed from table K. The same is true for the

$$x_1x_2 + x_1x_1'x_2'x_3 + x_1'x_2x_3$$

Figure 2

$$x_1 \quad x_2$$

		0								
0	$\frac{1}{2}$	0	0	0	0	0	0	0	0	0
		$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0
	1	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
	1	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
1	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
	1	0	0	0	0	0	0	0	0	0
		0	$\frac{1}{2}$	1	0	$\frac{1}{2}$	1	0	$\frac{1}{2}$	1	x_4
			0			$\frac{1}{2}$			1		x_3

$$x_1 x_2 x_2' x_3 x_4' + x_1' x_2 x_2' x_3 x_4' + x_1' x_2 x_2' x_3' x_4$$

$$+ x_1' x_2 x_2' x_3' x_4' + x_1 x_2 x_2' x_3 x_4$$

Figure 3

second two rows of K and the second row of K_1, so the second two rows of K are also removed. The phrase represented by the third row of K_1, created from the last row of K and the first row of K_1, is subsumed by the last phrase of F_3, so the last row is removed from K. The three rows of K_1, which cannot create any new rows, and whose phrases do not subsume one another, represent the function in minimized form, which is the same form computed in Example 3a.

Example 4:

$$F_4(x_1,x_2,x_3,x_4) = x_1 x_1' x_2' x_3 + x_2' x_3 x_4 x_4'$$
$$+ x_1 x_3' x_4 + x_1' x_2' x_4' + x_1 x_2' x_4$$

The GLD for this function is created in the same manner as in the first three examples (figure 5). We will only note here that the function is 1 only when one of the last three product terms is 1; that is, whenever x_1 and x_4 equal 1 and x_3 equals 0, or when $x_1, x_2,$ and x_4 all equal 0, or when x_1 and x_4 equal 1 and x_2 equals 0. Minimization of this function yields

$$F_4 = 1 \cdot X_1(1,1)$$
$$X_3(0,0)X_4(1,1) + 1 \cdot X_1(0,0)X_2(0,0)X_4(0,0) + 1 \cdot X_1(1,1)$$
$$X_2(0,0)X_4(1,1) + \tfrac{1}{2} \cdot X_1(0,\tfrac{1}{2})X_2(0,\tfrac{1}{2})X_4(0,\tfrac{1}{2}) + \tfrac{1}{2} \cdot X_1(\tfrac{1}{2},1)$$
$$X_2(0,\tfrac{1}{2})X_4(\tfrac{1}{2},1) + \tfrac{1}{2} \cdot X_1(\tfrac{1}{2},1)X_3(0,\tfrac{1}{2})X_4(\tfrac{1}{2},1). \text{ Using}$$

Part C(1) of the algorithm, the first three terms translate back to $x_1 x_3' x_4 + x_1' x_2' x_4' + x_1 x_2' x_4$. The fourth term, by Proposition 3, is covered by the second term. Likewise, the fifth and sixth terms are covered by the third and fourth terms, respectively. Therefore, F_4 is reduced to $x_1 x_3' x_4 + x_1' x_2' x_4' + x_1 x_2' x_4$, which is the result obtained by fuzzy minimization.

Example 5:

$$F_5(x_1,x_2,x_3,x_4) = x_1 x_1' x_2' x_3' + x_1 x_1' x_3 x_4' +$$
$$x_1 x_1' x_2 x_4$$
$$F_6(x_1,x_2,x_3,x_4) = x_1 x_1' x_3' x_4 + x_1 x_1' x_2 x_3 +$$
$$x_1 x_1' x_2' x_4'$$

As it can be seen from figures 6 and 7, the GLD's for these FSF's are identical. Also, both F_5 and F_6 are already in minimized form. Figure 6 yields $\tfrac{1}{2} \cdot X_1(\tfrac{1}{2},\tfrac{1}{2})X_2(0,\tfrac{1}{2})X_3(0,\tfrac{1}{2}) + \tfrac{1}{2} \cdot X_1(\tfrac{1}{2},\tfrac{1}{2})X_3(\tfrac{1}{2},1)$

$X_4(0,\tfrac{1}{2}) + \tfrac{1}{2} \cdot X_1(\tfrac{1}{2},\tfrac{1}{2})X_2(\tfrac{1}{2},1)X_4(\tfrac{1}{2},1)$. This translates to the orginal F_5. In a similar manner, figure 7

269

$$F_3(x_1, x_2, x_3, x_4) \rightarrow 10111001 + 01111001 + 01111001$$
$$+ 01110101 + 10111010$$

$$F_3(x_1, x_2, x_3, x_4) \rightarrow 2,3,2,1 + 1,3,2,1 + 1,3,2,1$$
$$+ 1,3,1,1 + 2,3,2,2$$

x_1	x_2	x_3	x_4	
2	3	2	1	x
1	3	2	1	x
1	3	1	2	x
1	3	1	1	x
2	3	2	2	x

K =

x_1	x_2	x_3	x_4
0	3	2	1
1	3	1	0
2	3	2	0

$K_1 =$

$$F_3(x_1, x_2, x_3, x_4) \rightarrow 0,3,2,1 + 1,3,1,0 + 2,3,2,0$$

$$F_3(x_1, x_2, x_3, x_4) \rightarrow 00111001 + 01110100 + 10111000$$

$$F_3(x_1, x_2, x_3, x_4) = x_2 x_2' x_3 x_4' + x_1' x_2 x_2' x_3' + x_1 x_2 x_2' x_3$$

Figure 4

yields a function which translates back to the orginal F_6.

Therefore, we have two minimized forms of a FSF. We see that the proposed algorithm gives us an alternative means of identifying equivalent minimized FSF's, since if they are equivalent, they should have identical representations in a GLD.

4. CONCLUSION

We have looked at some relationships between fuzzy switching functions and multi-valued switching functions. Although their basic concepts are different, they have many similar properties. In fact, since Marinos' fuzzy classification (5) allows us to partition the range of fuzzy values into any number of discrete classes, we can say that a multi-valued switching function is actually just one type of fuzzy switching function.

In examining the relationships between FSF's and MVSF's, we found that FSF's can be minimized using methods developed for MVSF's. An algorithm for such a minimization was presented in Chapter 3.

Several examples of fuzzy minimization using this algorithm were presented in Chapter 3, along with an example of fuzzy minimization using the fuzzy iterative concensus, namely, the algorithm

presented by Kandel and Francioni.(4). The two algorithms differ in several ways. For instance, in the latter algorithm, each term must be checked to see if it subsumes any other term in the function. This is not done in the algorithm presented in this paper. Also, in the algorithm in (4), each term must be compared with every other term to see if any new terms can be created; then we must determine if any old terms can be discarded. This process is iteratively repeated (possibly several times) until the function is completely minimized. In the algorithm in this paper, each term is examined at most twice; once to establish the 1's in the table, and once to establish the ½'s.

Of course, the algorithm in this paper requires that all values of the function must be computed from inputs from $(0, ½, 1)^n$, while no computation of values is necessary in the algorithm in (4).

These are some qualities to be considered when choosing a fuzzy minimization algorithm. What is now needed is a comparison of the complexity of the fuzzy minimization techniques presented over the past few years.

$$x_1x_1'x_2'x_3 + x_2'x_3x_4x_4' + x_1x_3'x_4 + x_1'x_2'x_4' + x_1x_2'x_4$$

Figure 5

$$x_1x_1'x_2'x_3' + x_1x_1'x_3x_4' + x_1'x_1'x_2x_4$$

Figure 6

271

x_1	x_2									
	0	0	0	0	0	0	0	0	0	0
0	$\frac{1}{2}$	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0
	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
	1	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
	0	0	0	0	0	0	0	0	0	0
1	$\frac{1}{2}$	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0
		0	$\frac{1}{2}$	1	0	$\frac{1}{2}$	1	0	$\frac{1}{2}$	1	x_4
		0			$\frac{1}{2}$			1			x_3

$$x_1 x_1' x_3' x_4 \;+\; x_1 x_1' x_2 x_3 \;+\; x_1 x_1' x_2' x_4'$$

Figure 7

REFERENCES

(1) Allen, C.M. and Givone, D.D. "The Allen-Givone Implementation Algebra." In Computer Science and Multiple-Valued Logic, pp. 262 - 282. Edited by D.C. Rine, Amsterdam: North Holland, 1977.

(2) Kandel, A. and Yager, R.R. "A 1979 Bibliography on Fuzzy Sets, Their Applications, and Related Topics." In Advances in Fuzzy Set Theory and Applications, pp. 621 - 744. Edited by M. M. Gupta, R. K. Ragade, and R. R. Yager. Amsterdam: North Holland, 1979.

(3) Kandel, A. and Byatt, W. J. "Fuzzy Sets, Fuzzy Algebra, and Fuzzy Statistics." Proceedings of the IEEE 66 (December 1978): 1619 - 1639.

(4) Kandel, A. and Francioni, J. "On the Properties and Applications of Fuzzy-Valued Switching Functions." IEEE Transactions on Computers C-29 (November 1980): 986-994.

(5) Marinos, P. N. "Fuzzy Logic and its Application to Switching Systems." IEEE Transactions on Computers C-18 (April 1969): 343-347.

(6) Michalski, R. S. "A Geometrical Model for the Synthesis of Interval Covers." University of Illinois, Department of Computer Science, Report No. 461 (June 24, 1971).

(7) Rescher, N. Many-Valued Logic. New York: McGraw-Hill Book Co., 1969.

(8) Thum, M. R. "The Complexity of Growth of Fuzzy Switching Functions Depending on the Number of Variables.' Master's Thesis Florida State University, Department of Mathematics and Computer Science, 1981.

(9) Zadeh, L. A. "Fuzzy Sets", Information and Control 8 (June 1965): 338-353.

M. Mizumoto

Department of Management Engineering
Osaka Electro-Communication University
Neyagawa, Osaka 572, Japan

This paper indicates that most of fuzzy translating rules for a fuzzy conditional proposition "If x is A then y is B" with A and B being fuzzy concepts can infer very reasonable consequences which fit our intuition with respect to several criteria such as modus ponens and modus tollens, if new compositions called "max-⊙ composition" and "max-∧ composition" are used in the compositional rule of inference, though, as was pointed out previously, reasonable consequences can not always be obtained when using the max-min composition which is used usually in the compositional rule of inference.

INTRODUCTION

In our daily life we often make such an inference of the form:

    Ant 1: If x is A then y is B
    Ant 2: x is A'
    ─────────────────────────────
    Cons: y is B'

where A, A', B, B' are fuzzy concepts. In order to make such an inference with fuzzy concepts, Zadeh /1/ suggested an inference rule called "compositional rule of inference" which infers B' of Cons from Ant 1 and Ant 2 by taking the max-min composition of fuzzy set A' and the fuzzy relation which is translated from the fuzzy conditional proposition "If x is A then y is B." In this connection, he /1/, Mamdani /2/ and Mizumoto /3,4/ suggested several translating rules for translating the fuzzy proposition "If x is A then y is B" into a fuzzy relation.

In /3,4/ we pointed out that the consequences inferred by Zadeh's and Mamdani's translating rules do not always fit our intuition, and proposed some new translating rules which can get the consequences coinciding with our intuition with respect to several criteria such as modus ponens and modus tollens. Moreover, we suggested in /4/ new translating rules which are obtained by introducing implication rules of many-valued logic systems, but these translating rules were found not to infer reasonable consequences.

In /5/ we have shown that, althogh the translating rule by Zadeh called "arithmetic rule" does not infer reasonable consequences in the compositional rule of inference which uses the max-min composition, the arithmetic rule can infer very reasonable consequences when new compositions named "max-⊙ composition" and "max-∧ composition" are used in the compositional rule of inference, where ⊙ is the operation of "bounded-product" which is dual to "bounded-sum" introduced by Zadeh /1/, and ∧ is the operation of "drastic product" $T_w(x,y)$ introduced by Dubois /6/.

As the continuation of our study /5/, this paper investigates the inference results by all the translating rules proposed until now under the max-⊙ composition and max-∧ composition, and shows that the majority of the translating rules can infer very reasonable consequences which fit our intuition.

TRANSLATING RULES

We shall first consider the following form of inference in which a fuzzy conditional proposition is contained.

$$\begin{array}{ll} \text{Ant 1:} & \text{If } x \text{ is } A \text{ then } y \text{ is } B \\ \text{Ant 2:} & x \text{ is } A' \\ \hline \text{Cons:} & y \text{ is } B' \end{array} \quad (1)$$

where x and y are the names of objects, and A, A', B and B' are fuzzy concepts represented by fuzzy sets in universes of discourse U, U, V and V, respectively. This form of inference may be viewed as fuzzy modus ponens which reduces to the classical modus ponens when A' = A and B' = B.

Moreover, the following form of inference is possible which contains a fuzzy conditional proposition.

$$\begin{array}{ll} \text{Ant 1:} & \text{If } x \text{ is } A \text{ then } y \text{ is } B \\ \text{Ant 2:} & y \text{ is } B' \\ \hline \text{Cons:} & x \text{ is } A' \end{array} \quad (2)$$

This inference can be considered as fuzzy modus tollens which reduces to the classical modus tollens when B' = not B and A' = not A.

The fuzzy proposition "If x is A then y is B" of (1) and (2) represents a certain relationship between A and B. From this point of view, a number of translating rules were proposed for translating the fuzzy proposition "If x is A then y is B" into a fuzzy relation in U x V.

Let A and B be fuzzy sets in U and V, respectively, and let x, ∪, ∩, ¬ and ⊕ be cartesian product, union, intersection, complement and bounded-sum for fuzzy sets. Then the following fuzzy relations in U x V are

273

obtained from the fuzzy proposition "If x is A then y is B". Rm (maximin rule) and Ra (arithmetic rule) were proposed by Zadeh /1/, Rc (min rule) by Mamdani /2/, and the other were by Mizumoto et al. /3,4/ by introducing the implications of many-valued logic systems.

$$R_m = (A \times B) \cup (\neg A \times V)$$

$$\Leftrightarrow (\mu_A(u) \wedge \mu_B(v)) \vee (1-\mu_A(u)) \qquad (3)$$

$$R_a = (\neg A \times V) \oplus (U \times B)$$

$$\Leftrightarrow 1 \wedge (1-\mu_A(u)+\mu_B(v)) \qquad (4)$$

$$R_c = A \times B$$

$$\Leftrightarrow \mu_A(u) \wedge \mu_B(v) \qquad (5)$$

$$R_s = A \times V \underset{s}{\Rightarrow} U \times B$$

$$\Leftrightarrow \begin{cases} 1 & \dots \ \mu_A(u) \leqq \mu_B(v), \\ 0 & \dots \ \mu_A(u) > \mu_B(v). \end{cases} \qquad (6)$$

$$R_g = A \times V \underset{g}{\Rightarrow} U \times B$$

$$\Leftrightarrow \begin{cases} 1 & \dots \ \mu_A(u) \leqq \mu_B(v), \\ \mu_B(v) & \dots \ \mu_A(u) > \mu_B(v). \end{cases} \qquad (7)$$

$$R_{sg} = (A \times V \underset{s}{\Rightarrow} U \times B) \cap (\neg A \times V \underset{g}{\Rightarrow} U \times \neg B) \qquad (8)$$

$$R_{gg} = (A \times V \underset{g}{\Rightarrow} U \times B) \cap (\neg A \times V \underset{g}{\Rightarrow} U \times \neg B) \qquad (9)$$

$$R_{gs} = (A \times V \underset{g}{\Rightarrow} U \times B) \cap (\neg A \times V \underset{s}{\Rightarrow} U \times \neg B) \qquad (10)$$

$$R_{ss} = (A \times V \underset{s}{\Rightarrow} U \times B) \cap (\neg A \times V \underset{s}{\Rightarrow} U \times \neg B) \qquad (11)$$

$$R_b = (\neg A \times V) \cup (U \times B)$$

$$\Leftrightarrow (1-\mu_A(u)) \vee \mu_B(v) \qquad (12)$$

$$R_\Delta = A \times V \underset{\Delta}{\Rightarrow} U \times B$$

$$\Leftrightarrow \begin{cases} 1 & \dots \ \mu_A(u) \leqq \mu_B(v), \\ \dfrac{\mu_B(v)}{\mu_A(u)} & \dots \ \mu_A(u) > \mu_B(v). \end{cases} \qquad (13)$$

$$R_\blacktriangle = A \times V \underset{\blacktriangle}{\Rightarrow} U \times B$$

$$\Leftrightarrow \begin{cases} 1 \wedge \dfrac{\mu_B(v)}{\mu_A(u)} \wedge \dfrac{1-\mu_A(u)}{1-\mu_B(v)} & \dots \ \mu_A(u) > 0, 1-\mu_B(v) > 0, \\ 1 & \dots \ \mu_A(u) = 0 \text{ or } 1-\mu_B(v) = 0. \end{cases} \qquad (14)$$

$$R_\star = A \times V \underset{\star}{\Rightarrow} U \times B$$

$$\Leftrightarrow 1 \wedge \mu_A(u) + \mu_A(u)\mu_B(v). \qquad (15)$$

$$R_\# = A \times V \underset{\#}{\Rightarrow} U \times B \qquad (16)$$

$$\Leftrightarrow (1-\mu_A(u) \vee \mu_B(v)) \wedge ((\mu_A(u) \vee 1-\mu_A(u)) \wedge (\mu_B(v) \vee 1-\mu_B(v)).$$

$$R_\square = A \times V \underset{\square}{\Rightarrow} U \times B$$

$$\Leftrightarrow \begin{cases} 1 & \dots \ \mu_A(u) < 1 \text{ or } \mu_B(v) = 1 \\ 0 & \dots \ \mu_A(u) = 1, \mu_B(v) < 1 \end{cases} \qquad (17)$$

In the fuzzy modus ponens of (1), the consequence B' can be deduced from Ant 1 and Ant 2 by taking the max-min composition "o" of the fuzzy set A' and the fuzzy relation obtained above (the compositional rule of inference). For example, we have for the translating rule Rm of (3)

$$B_m' = A' \text{ o } R_m \qquad (18)$$

$$= A' \text{ o } \big[(A \times B) \cup (\neg A \times V)\big]$$

The membership function of the fuzzy set Bm' in V is given as

$$\mu_{B_m'}(v) = \underset{u}{V}\big\{\mu_{A'}(u) \wedge \mu_{R_m}(u,v)\big\} \qquad (19)$$

$$= \underset{u}{V}\big\{\mu_{A'}(u) \wedge [(\mu_A(u) \wedge \mu_B(v)) \vee (1-\mu_A(u))]\big\}$$

Similarly, in the case of fuzzy modus tollens of (2), the consequence A' is given by

$$A_m' = R_m \text{ o } B' \qquad (20)$$

As simple examples, let A' = A in (18) and B' = not B in (20), then we can have such inference results /4/ as

$$B_m' = A \text{ o } R_m \Leftrightarrow \mu_{B_m'}(v) = 0.5 \vee \mu_B(v)$$

$$A_m' = R_m \text{ o } \underline{not} \ B \Leftrightarrow \mu_{A_m'}(u) = 0.5 \vee (1-\mu_A(u))$$

Similarly, for the arithmetic rule of (4)

$$B_a' = A \text{ o } R_a \Leftrightarrow \mu_{B_a'}(v) = \frac{1 + \mu_B(v)}{2}$$

$$A_a' = R_a \text{ o } \underline{not} \ B \Leftrightarrow \mu_{A_a'}(u) = \frac{1 - \mu_A(u)}{2}$$

These consequences B' and A' are found not to be equal to B and not A, respectively. In other words, these translating rules can not satisfy the modus ponens and modus tollens which are quite reasonable demands in the fuzzy conditional inference. Therefore, it seems that these rules are not suitable

> If x is A then y is B
> x is A _____ (modus ponens) (21)
> y is B

> If x is A then y is B
> y is not B _____ (modus tollens) (22)
> x is not A

methods for the fuzzy conditional inference. In the next section, however, we shall show that not only these rules but also other translating rules in (5)-(16) can satisfy the modus ponens and modus tollens and infer the consequences which fit our intuition, if, instead of the max-min composition usually used in the compositional rule of inference, we use two kinds of new compositions called "max-⊙ composition" and "max-∧ composition" in the compositional rule of inference.

FUZZY CONDITIONAL INFERENCE UNDER NEW COMPOSITIONS

We shall first give the operations of "bounded-product" ⊙ and "drastic product" ∧ in order to define new compositions of "max-

θ composition" and "max-∧ composition" to be used in the compositional rule of inference. The more detailed properties of these operations are found in /6-8/. For any $x, y \in [0,1]$

Bounded-Product: $\quad x \ominus y = 0 \vee (x + y - 1)$ (23)

Drastic Product:
$$x \wedge y = \begin{cases} x & \cdots & y = 1 \\ y & \cdots & x = 1 \\ 0 & \cdots & x, y < 1 \end{cases} \quad (24)$$

Using these new operations we can easily define new operations called max-θ composition "□" and max-∧ composition "▲" in the same way as (18) and (20).

$$Bm' = A' \; \square \; Rm \quad (25)$$
$$\Leftrightarrow \mu_{Bm'}(v) = \bigvee_u \left\{ \mu_{A'}(u) \ominus \mu_{Rm}(u,v) \right\}$$

$$Am' = Rm \; \square \; B' \quad (26)$$

Similarly, under the max-∧ composition "▲", we have

$$Bm' = A' \; \blacktriangle \; Rm \quad (27)$$
$$\Leftrightarrow \mu_{Bm'}(v) = \bigvee_u \left\{ \mu_{A'}(u) \wedge \mu_{Rm}(u,v) \right\}$$

$$Am' = Rm \; \blacktriangle \; B' \quad (28)$$

The same ways are applicable to other translating rules Ra, Rc, ..., R□ of (4)-(17).
In the fuzzy modus ponens, we shall show what the consequences B' become under new compositions "□" and "▲" when A' is

$A' = A$
$A' = \underline{very} \; A = A^2$
$A' = \underline{more \; or \; less} \; A = A^{0.5}$
$A' = \underline{not} \; A = \neg A$

which are typical examples of A'.
Similarly, in the fuzzy modus tollens of (2), we show what the consequences A' is when B' is

$B' = \underline{not} \; B = \neg B$
$B' = \underline{not \; very} \; B = \neg B^2$
$B' = \underline{not \; more \; or \; less} \; B = \neg B^{0.5}$
$B' = \overline{B}$

We shall begin with the fuzzy modus ponens in (1). It is assumed in the discussion of the fuzzy modus ponens that $\mu_A(u)$ takes all values in $[0,1]$ according to u varying all over U, that is, μ_A is a function onto $[0,1]$. Clearly, from this assumption, the fuzzy set A is a normal fuzzy set.
We shall first discuss Rm and obtain the consequence Bm' of (25) at $A' = A^{\alpha}$ which is a general case of A, $\underline{very}$ A and $\underline{more \; or \; less}$ A. From the above assumption that μ_A is a function onto $[0,1]$, (25) can be rewritten as

$$bm' = \bigvee_x \left\{ x^{\alpha} \ominus \left[(x \wedge b) \vee (1-x) \right] \right\} \quad (29)$$
$$f(x) = x^{\alpha} \ominus \left[(x \wedge b) \vee (1-x) \right] \quad (30)$$

by letting

$$\mu_A(u) = x, \quad \mu_B(v) = b, \quad \mu_{Bm'}(v) = bm' \quad (31)$$

From the definition of bounded-product of (23), we have f(x) of (30) as

$f(x)$
$$= 0 \vee \left\{ x^{\alpha} + \left[(x \wedge b) \vee (1-x) \right] - 1 \right\}$$
$$= 0 \vee \left[(x^{\alpha} + x - 1) \wedge (x^{\alpha} + b - 1) \right] \vee (x^{\alpha} - x) \quad (32)$$

Case of $\alpha \gtrless 1$: When $\alpha \gtrless 1$, $x^{\alpha} - x \lesssim 0$ is obtained. Thus, f(x) reduces to

$$f(x) = 0 \vee \left[(x^{\alpha} + x - 1) \wedge (x^{\alpha} + b - 1) \right]$$

Fig.1(a) shows partly the expressions $x^{\alpha} + x - 1$ and $x^{\alpha} + b - 1$ by using a parameter b. When b is equal to, say, 0.2, f(x) is indicated by the broken line and thus bm' = $\bigvee_x$ f(x) of (29) at b = 0.2 becomes 0.2 by taking the maximum of this line. In the same way, at b = 0.7, f(x) is shown by the dot-dash line whose maximum value is 0.7. Thus we have bm' = 0.7 at b = 0.7. In general, we can have bm' = b for any b, that is, bm' = b at $x' = x^{\alpha}$ ($\alpha \gtrless 1$), which leads to $\mu_{Bm'}(v) = \mu_B(v)$ at $\mu_{A'}(u) = \mu_A(u)^{\alpha}$ ($\alpha \gtrless 1$) from (31). Namely, Bm' = B at $A' = A^{\alpha}$ ($\alpha \gtrless 1$). Therefore,

$$A^{\alpha} \; \square \; Rm = B \quad \cdots \; \alpha \gtrless 1 \quad (33)$$
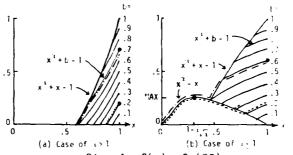
Fig. 1   f(x) of (32)

Case of $\alpha \lesssim 1$: f(x) is given by (32) and is drawn in Fig.1(b). The expression $x^{\alpha} - x$ ($\alpha < 1$) has the maximum value $^{1-\alpha}\!\!\sqrt{\alpha}(\frac{1}{\alpha} - 1)$ (= MAX) at $x = {^{1-\alpha}\!\!\sqrt{\alpha}}$. Fig1.(b) indicates that bm' = MAX at $0 \lesssim b \lesssim$ MAX, that is,

$$bm' = \bigvee_x f(x) = {^{1-\alpha}\!\!\sqrt{\alpha}}(\frac{1}{\alpha} - 1)$$

On the other hand, when MAX $\lesssim$ b $\lesssim$ 1, we have

$$bm' = b.$$

Hence,

$$bm' = {^{1-\alpha}\!\!\sqrt{\alpha}}(\frac{1}{\alpha} - 1) \vee b$$

Namely,

$$\mu_{Bm'}(v) = {^{1-\alpha}\!\!\sqrt{\alpha}}(\frac{1}{\alpha} - 1) \vee \mu_B(v) \quad at \; \alpha \lesssim 1. \quad (34)$$

Therefore, the consequence Bm' = $A^{\alpha} \; \square \; Rm$ under the max-θ composition "□" is given by

$$\mu_{Bm'}(v) = \begin{cases} {^{1-\alpha}\!\!\sqrt{\alpha}}(\frac{1}{\alpha} - 1) \vee \mu_B(v) & \cdots \alpha \lesssim 1 \\ \mu_B(v) & \cdots \alpha \gtrless 1 \end{cases} \quad (35)$$

From this result we can obtain the consequences Bm' at $A' = A$, $\underline{very}$ A (= A^2) and $\underline{more \; or \; less}$ A (= $A^{0.5}$) by letting α = 1, 2, 0.5.

275

Table 1

	A	very A	more or less A	not A
Rm	B	B	$\frac{1}{4} \vee \mu_B$	unknown
Ra	B	B	$\begin{cases} \mu_B + \frac{1}{4} \cdots \mu_B \leqq \frac{1}{4} \\ \sqrt{\mu_B} \cdots \mu_B \geqq \frac{1}{4} \end{cases}$	unknown
Rc	B	B	B	∅
Rs	B	very B	more or less B	unknown
Rg	B	B	more or less B	unknown
Rsg	B	very B	more or less B	not B
Rgg	B	B	more or less B	not B
Rgs	B	B	more or less B	not B
Rss	B	very B	more or less B	not B
Rb	B	B	$\frac{1}{4} \vee \mu_B$	unknown
R_Δ	B	B	more or less B	unknown
$R_\blacktriangle$	B	very B	more or less B	unknown
R_*	B	B	$\begin{cases} \frac{1}{4(1-\mu_B)} \cdots \mu_B \leqq \frac{1}{2} \\ \mu_B \cdots \mu_B \geqq \frac{1}{2} \end{cases}$	unknown
$R_{\divideontimes}$	b	B	$\frac{1}{4} \vee \mu_B$	B ∪ not B
$R_\square$	unknown	unknown	unknown	unknown

Table 2

	not B	not very B	more or less not B	B
Rm	not A	$(1-\mu_A) \vee \frac{1}{4}$	not A	A ∪ not A
Ra	not A	$\begin{cases} 1-\mu_A^2 \cdots \mu_A \leqq \frac{1}{2} \\ \frac{1}{4}+(1-\mu_A) \cdots \mu_A \geqq \frac{1}{2} \end{cases}$	not A	unknown
Rc	∅	$\begin{cases} \mu_A-\mu_A^2 \cdots \mu_A \leqq \frac{1}{2} \\ \frac{1}{4} \cdots \mu_A \geqq \frac{1}{2} \end{cases}$	∅	A
Rs	not A	not very A	not more or less A	unknown
Rg	not A	$(1-\mu_A^2) \vee \frac{1}{4}$	not more or less A	unknown
Rsg	not A	not very A	not more or less A	A
Rgg	not A	$(1-\mu_A^2) \vee \frac{1}{4}$	not more or less A	A
Rgs	not A	$(1-\mu_A^2) \vee \frac{1}{4}$	not more or less A	A
Rss	not A	not very A	not more or less A	A
Rb	not A	$(1-\mu_A) \vee \frac{1}{4}$	not A	unknown
P_Δ	not A	$\begin{cases} 1-\mu_A^2 \cdots \mu_A \leqq \frac{\sqrt{2}}{2} \\ \frac{1}{4\mu_A^2} \cdots \mu_A \geqq \frac{\sqrt{2}}{2} \end{cases}$	not more or less A	unknown
$R_\blacktriangle$	not A	not very A	not more or less A	unknown
R_*	not A	$(1 - \frac{\mu_A}{2})^2$	not A	unknown
$R_{\divideontimes}$	not A	$(1-\mu_A) \vee \frac{1}{4}$	not A	A ∪ not A
$R_\square$	$\begin{cases} 1 \cdots \mu_A<1 \\ 0 \cdots \mu_A=1 \end{cases}$	$\begin{cases} 1 \cdots \mu_A<1 \\ 0 \cdots \mu_A=1 \end{cases}$	$\begin{cases} 1 \cdots \mu_A<1 \\ 0 \cdots \mu_A=1 \end{cases}$	unknown

$$Bm' = A \square Rm = B \qquad (36)$$
$$Bm' = \underline{\text{very}} \ A \square Rm = B \qquad (37)$$
$$Bm' = \underline{\text{more or less}} \ A \square Rm$$
$$= \frac{1}{4} \vee \mu_B(v) \qquad (38)$$

(36) indicates that a modus ponens is satisfied by the method Rm under the max-θ composition "□". It is noted that Rm does not satisfy the modus ponens under the max-min composition "o".

We shall next consider the inference result $Ba' = A^\alpha \blacktriangle Ra$ under the max-∧ composition "▲". Ba' is given by

$$\mu_{Ba'}(v) = \bigvee_u \{ \mu_{A^\alpha}(u) \wedge [1 \wedge (1 - \mu_A(u) + \mu_B(v))] \}$$

$$ba' = \bigvee_x \{ x^\alpha \wedge [1 \wedge (1 - x + b)] \}$$

Let $g(x)$ be

$$g(x) = x^\alpha \wedge [1 \wedge (1 - x + b)] \qquad (39)$$

then $g(x)$ is shown by the solid line and the black circle in Fig.2. Namely,

Fig.2 g(x) of (39)

$$g(x) = \begin{cases} x^\alpha & \cdots & 0 \leqq x \leqq b \\ b & \cdots & x = 1 \\ 0 & \cdots & \text{otherwise} \end{cases}$$

Thus,

$$ba' = \bigvee_x g(x) = \bigvee_x x^\alpha \vee b = b^\alpha \vee b.$$

Therefore, we have $Ba' = A^\alpha \blacktriangle Ra$ as

$$Ba' = \begin{cases} B^\alpha & \cdots \alpha \leqq 1 \\ B & \cdots \alpha \geqq 1 \end{cases} \qquad (40)$$

which gives the inference results Ba' at A' = A, <u>very</u> A, and <u>more or less</u> A as follows.

$$Ba' = A \blacktriangle Ra = B, \qquad (41)$$
$$Ba' = \underline{\text{very}} \ A \blacktriangle Ra = B, \qquad (42)$$
$$Ba' = \underline{\text{more or less}} \ A \blacktriangle Ra$$
$$= \underline{\text{more or less}} \ B \qquad (43)$$

which also indicate the satisfaction of the modus ponens under the max-∧ composition "▲".

In the same way, we can obtain the consequences by other methods Rc, Rs, ..., R□. Tables 1-4 list the inference results by all the translating rules (3)-(17) under the max-θ composition "□" and max-∧ composition "▲".

276

Table 3 Inference Results under Max-∧ Composition "▲"
(Case of Fuzzy Modus Ponens)

	A	very A	more or less A	not A
Rm	B	B	B	unknown
Ra	B	B	more or less B	unknown
Rc	B	B	b	∅
Rs	B	very B	more or less B	unknown
Rg	B	B	more or less B	unknown
Rsg	B	very B	more or less B	not B
Rgg	B	B	more or less B	not B
Rgs	B	B	more or less B	not B
Rss	B	very B	more or less B	not B
Rb	B	B	B	unknown
R△	B	B	more or less B	unknown
R▲	B	very B	more or less B	unknown
R✶	B	B	B	unknown
R❋	B	B	B	B ∪ not B
Rᴅ	unknown	unknown	unknown	unknown

Table 4 Inference Results under Max-∧ Composition "▲"
(Case of Fuzzy Modus Tollens)

	not B	not very B	not more or less B	B
Rm	not A	not A	not A	A ∪ not A
Ra	not A	not very A	not A	unknown
Rc	∅	∅	∅	A
Rs	not A	not very A	not more or less A	unknown
Rg	not A	not very A	not more or less A	unknown
Rsg	not A	not very A	not more or less A	A
Rgg	not A	not very A	not more or less A	A
Rgs	not A	not very A	not more or less A	A
Rss	not A	not very A	not more or less A	A
Rb	not A	not A	not A	unknown
R△	not A	not very A	not more or less A	unknown
R▲	not A	not very A	not more or less A	unknown
R✶	not A	not A	not A	unknown
R❋	not A	not A	not A	A ∪ not A
Rᴅ	$\begin{cases} 1 .. \mu_A < 1 \\ 0 .. \mu_A = 1 \end{cases}$	$\begin{cases} 1 .. \mu_A < 1 \\ 0 .. \mu_A = 1 \end{cases}$	$\begin{cases} 1 .. \mu_A < 1 \\ 0 .. \mu_A = 1 \end{cases}$	unknown

In the form of fuzzy conditional infer-
ences (1) and (2), it seems according to our
intuition that criteria between A' of Ant 2
and B' of Cons of the fuzzy modus ponens (1)
ought to be satisfied as shown in the left
part of Table 5 (cf. /3,4/). Similarly,
criteria for the fuzzy modus tollens (2) are
also shown in this table. The right part of
Table 5 indicates the satisfaction (0) or
failure (X) of each criterion by each trans-
lating rule by the use of the inference
results given in Tables 1-4. In order to
compare the inference results under the max-
Θ composition "□" and max-∧ composition "▲"
with those under the ordinal max-min compo-
sition "o", the satisfaction of each crite-
rion under the max-min composition is listed
in the table (cf. /4/).

From Tables 1-5 it follows that all the
inference methods except Rᴅ can satisfy so-
called modus ponens (21) under the max-Θ com-
position "□" and max-∧ composition "▲", but
only the methods Rc, Rs, ..., Rss can satisfy
the modus ponens under the max-min composi-
tion "o". The almost same holds for the
modus tollens of (22). Moreover, it is found
that majority of the translating rules can

infer very reasonable consequences under the
max-Θ composition and max-∧ composition,
though we cannot always get reasonable con-
sequences under the max-min composition as
shown in Table 5 /5/.

CONCLUSION

We have shown that, when the max-Θ
composition and max-∧ composition are used
in the compositional rule of inference, the
majority of fuzzy inference methods can get
very reasonable consequences which coincide
with our intuition with respect to several
criteria such as modus ponens and modus tol-
lens. It will be of interest to apply the
max-Θ composition and max-∧ composition to
fuzzy inferences which are of the more com-
plicated form such as

If x is A_1 then y is B_1 else
if x is A_2 then y is B_2 else
.
.
if x is A_n then y is B_n.
x is A'.
———————————————
y is B'.

These results will be presented in subsequent
papers.

Table 5 Satisfaction (O) or Failure (X) of Criterion for Each Method under Max-min Composition "o", Max-θ Composition "□" and Max-Δ Composition "▲"

Criterion	Ant 2	Cons	Rm	Ra	Rc	Rs	Rg	Rsg	Rgg	Rgs	Rss	Rb	RΔa	RΔ	Rg	Rb	RΔ
			o□▲	o□▲	o□▲	o□▲	o□▲	o□▲	o□▲	▲□▲	o□▲	□▲	□▲	□▲	□▲	□▲	□▲
I (modus ponens)	A	B	XOO	XOO	OOO	OOO	OOO	OOO	OOO	OOO	OOO	XOO	XOO	...	...	X.	XXX
II-1	very A	very B	XXX	XXX	XXX	OOO	XXX	OOO	XXX	XXX	OOO	XXX	XXX	X..	XXX	XXX	XXX
II-2	very A	B	XOO	XOO	OOO	XXX	OOO	XXX	OOO	OOO	XXX	XOO	XOO	XXX	X..	X..	XXX
III-1	more or less A	more or less B	XXX	XXO	XXX	OOO	OOO	OOO	OOO	OOO	OOO	XXX	XOO	XO	XXX	XAX	XXX
III-2	more or less A	B	XXO	XXX	OOO	XXX	XXX	XXX	XXX	XXX	XXX	XXO	XXX	XXX	XXO	XX.	AAX
IV-1	not A	unknown	OOO	OOO	XXX	OOO	OOO	XXX	XXX	XXX	XXX	OOO	OOO	OOO	OOO	XXX	OOO
IV-2	not A	not B	XXX	XXX	XXX	XXX	XXX	OOO	OOO	OOO	OOO	XXX	XXX	XXX	XXX	XXX	XXX
V (modus tollens)	not B	not A	XOO	XOO	XXX	OOO	XOO	OOO	XOO	XOO	OOO	XOO	XOO	XOO	XOO	XOO	XXX
VI-1	not very B	not very A	XXX	XXO	XXX	OOO	XXO	OOO	XXO	XXO	OOO	XXX	XXO	XOO	XXX	XXX	XXX
VI-2	not very B	not A	XXO	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXO	XXX	XXX	XXO	XXO	XXX
VII-1	not more or less B	not more or less A	XXX	XXX	XXX	OOO	XOO	OOO	XOO	XOO	OOO	XXX	XOO	XOO	XXX	XXX	XXX
VII-2	not more or less B	not A	XOO	XOO	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XOO	XXX	XXX	XOO	XOO	XXX
VIII-1	B	unknown	XXX	OOO	XXX	OOO	OOO	XXX	XXX	XXX	XXX	OOO	OOO	OOO	OOO	XXX	OOO
VIII-2	B	A	XXX	XXX	OOO	XXX	XXX	XOO	XOO	OOO	OOO	XXX	XXX	XXX	XXX	XXX	XXX

REFERENCES

1. Zadeh, L.A., Calculus of fuzzy restriction, in Fuzzy Sets and Their Applications to Cognitive and Decision Processes (Zadeh, et al., Eds.), Academic Press, New York, 1975, pp. 1-39.
2. Mamdani, E.H., Application of fuzzy logic to approximate reasoning using linguistic systems, IEEE Trans. on Comp. c-26, 1182-1191 (1977).
3. Mizumoto, M., Fukami, S. & Tanaka, K., Some methods of fuzzy reasoning, in Advances in Fuzzy Set Theory and Applications (Gupta, M.M. et al., Eds.), North-Holland, Amsterdam, 1979, pp. 117-136.
4. Mizumoto, M. & Zimmermann, H.J., Comparison of fuzzy reasoning methods, Fuzzy Sets and Systems, 8, 3, 253-283 (1982).
5. Mizumoto, M., Note on the arithmetic rule by Zadeh for fuzzy conditional inference, Cybernetics and Systems, 12, 3, 247-306, (1981).
6. Dubois, D., Quelques classes d'operateurs remarquables pour combiner des ensembles, Busefal, Automne, 29-35 (1979).
7. Mizumoto, M. & Tanaka, K., Fuzzy sets and their operations, I, Information and Control, 48, 30-48 (1981).
8. Mizumoto, M., Fuzzy sets and their operations, II, Information and Control, 50, 160-174 (1981).

A STUDY OF FUZZY RELATIONS AND THEIR INVERSE PROBLEM

Masaki Togai[*] and Paul P. Wang

Department of Electrical Engineering
Duke University
Durham, NC 27706
USA

Abstract

his paper consists of two main topics relating to the fuzzy logic
.oning: fuzzy implication and its inverse problem.

rstly, the characteristic difference of various types of implica-
inctions will be analyzed by introducing two new
s: namely the "joint" relation and the "conditional" one.

:condly, a novel method to find upper and lower bounds of the
of fuzzy inverse problem is introduced. In addition to the con-
il man-min operation, we propose a new operation, namely
iposition," to solve the problem effectively. Theorems concern-
bounds and the composite mappings of fuzzy sets are also
:d.

Introduction

his paper consists of two topics intimately related to the concept
y logic and reasoning - fuzzy implication and its inverse prob-

embi and Mamdani [1] and Mizumoto et al [2] have proposed
fuzzy implication functions in their papers. Some basic charac-
difference of implication function obtained through fuzzy
an product from those obtained by other means has brought to
ention while reviewing the papers mentioned above. It appears
e need exists to explain the differences on a more united basis;
we propose here the "joint" and "conditional" fuzzy relations.
o recognize that the concept of conditional and joint possibilities
:ated by Hisdel [3] turns to be very helpful in understanding the
is.

issuming that the conditional fuzzy relation is defined by the
mplication function, the canonical extension of Baye's formula
)ability theory to fuzzy set theory will be made in this paper.
arious way to handle implication in fuzz logic have been pro-
by many researchers [1,2], we feel it is necessary to compare
teristics of these implication functions.

irstly, the joint relational functions will be derived from the
it conditional functions or implication functions. Secondly, the
teristics of these relational functions associated with types of
ition will also be investigated in depth.

he relations mentioned above are, in a sense, the mapping from
iverse of discourse, say U, to another universe of discourse, say
fuzzy inverse problem, on the other hand, deals with a mapping
rse direction, i.e., from V to U, under the assumption that the
ig from U to V does exist.

A fuzzy inverse problem was first proposed by Sanchez [4]. He
investigated the problem to find $Q \subset X \times Y$ knowing $R \subset X \times Z$ and
$S \subset X \times Z$ such that

$$Q \circ R = S$$

He further showed a condit   of the solution by giving the least upper
bound of the solution. Tashiro et al. [5] and Tsukamoto[6,7] followed
Sanchez's work, and proposed an algorithm to find the lower bound of
the solution for the problem stated as follows "given a fuzzy relation
R on $U \times V$ and a fuzzy subset B of V, find all the fuzzy subsets A of
U such that $B = A \circ R$".

In the second half of this paper we propose a new method to
search a lower and upper bound of the solution for the fuzzy inverse
problem. First, we propose a simple operation to find the solution of
the inverse problem, then, we prove that the proposed operation gives
the necessary and sufficient range of the solution.

Fuzzy Sets and Relevant Properties

A fuzzy subset A of a universe of discourse U is the set of
ordered pairs defined by

$$A = \{(u, \mu_A(u)) | u \in U \text{ and } \mu_A \in [0,1]\}. \tag{1}$$

A fuzzy subset A can be viewed as the union of its constituent ele-
ments with characteristic function μ_A's. On the basis, A may be
represented in the form suggested by Zadeh [8]:

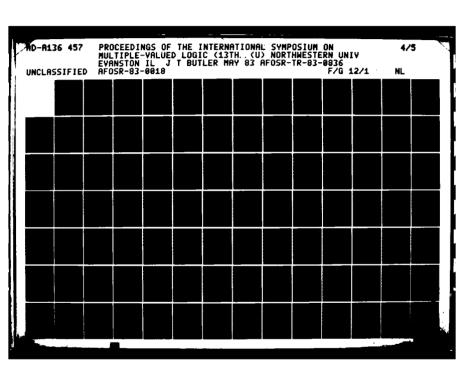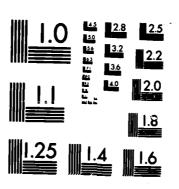$$A = \int_U \mu_A(u)/u \tag{2}$$

where the integral sign stands for the union of $\mu_A(u)/u$. If A has a
finite number of elements $u_1, u_2, ..., u_N$ in U, it may be alternatively
represented in the form

$$A = \mu_A(u_1)/u_1 + \mu_A(u_2)/u_2 + \cdots + \mu_A(u_N)/u_N .$$

or

$$A = \sum_{i=1}^{N} \mu_A(u_i)/u_i . \tag{3}$$

MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

Fuzzy Cartesian Product --- Let A be a fuzzy subset of a universe of discourse U, and let B be a fuzzy subset of a possibly different universe of discourse V. Then, the fuzzy Cartesian product of A and B, denoted by $A \times B$ symbolically, is defined by

$$A \underset{\sim}{\times} B = \int_{U \times V} \mu_A(u) \wedge \mu_B(v)/(u,v) \tag{4}$$

In other words, $A \times B$ is a fuzzy set of ordered pairs (u,v), $u \in U$, $v \in V$, with the grade of membership of (u,v) in $A \times B$ characterized by $\mu_A(u) \wedge \mu_B(v)$.

Domain and Range --- Let $R \subset U \times V$ be a fuzzy relation. The *domain* of R, denoted by dom(R), and the *range* of R, denoted by ran(R), are defined respectively by

$$\mu_{dom(R)}(u) = \underset{v}{V} \mu_R(u,v); \quad \forall u \in U \tag{5a}$$

and

$$\mu_{ran(R)}(v) = \underset{y}{V} \mu_R(u,v); \quad \forall v \in V. \tag{5b}$$

Subdomain and Subrange --- Let $R \subset U \times V$ be a fuzzy relation. The *subdomain* of R, denoted by sdom(R), and the subrange of R, denoted by sran(R), are defined by

$$\mu_{sdome(R)}(u) = \underset{v}{\wedge} \mu_R(u,v); \quad \forall u \in U \tag{6a}$$

and

$$\mu_{sran(R)}(v) = \underset{u}{\wedge} \mu_R(u,v); \quad \forall v \in V. \tag{6b}$$

Inversion or Transposition --- Let $R \subset U \times V$ be a fuzzy relation. The *inverse* or *transpose* of R, denoted by R^{-1} or R^T, is the fuzzy relation on $V \times U$ defined by

$$\mu_R(u,v) = \mu_{R^{-1}}(v,u) \tag{7}$$

$$= \mu_{R^T}(v,u).$$

Separability --- A relation R on $U \times V$ is said to be "separable" if and only if there exist $A \subset U$ and $B \subset V$ such that $R = A \times B$, where

$$A = Sup\{ u| uRv \} = dom(R) \tag{8a}$$

$$B = Sup\{ v| uRv \} = ran(R). \tag{8b}$$

Noninteraction --- A and B are "noninteractive" under $R = A \times B$ if and only if R is separable. This concept is analogous to the well known concept so called "independence" of random variables in probability theory.

Fuzzy Implications and Their Impact on Fuzzy Relations

In this section we will present the basic definitions of conditional fuzzy relations and joint fuzzy relations and hence demonstrate how a marginal membership function can be induced from the joint membership function. Some insight concerning the interplay between these two relations will be of some interest. As a result of our investigation, the justification of the use of the max-min operation for the compositional rule of inference as defined by Zadeh [8] may be explained in a more satisfactory manner.

There are a number of ways that the conditional membership function can be constructed. We intend to summarize here the most frequently encountered types of implications in fuzzy logic to demonstrate their potential usefulness for future applications. The effect of an implication to fuzzy relations will also be investigated accordingly.

Conditional Fuzzy Relation --- A relation from a subset A to another subset B, designated by the fuzzy implication function, is a conditional relation and symbolically denoted as $R_{A \rightarrow B}$. This relation is characterized by a bivariate membership function:

$$R_{A \rightarrow B} = \int_{U \times V} \mu_{R_{A \rightarrow B}}(v/u)/(u,v) \tag{9}$$

where $A \subset U$, $B \subset V$, $u \in V$, and $v \in V$.

Joint Fuzzy Relation --- The joint fuzzy relation $R_{A,B} \subset U \times V$ is defined by the membership function $\mu_{R_{A,B}}(u,v)$ satisfying

$$\mu_{R_{A,B}}(u,v) = \mu_A(u) \wedge \mu_{R_{A \rightarrow B}}(v/u) \tag{10}$$

where $A \subset U$, $B \subset V$, $u \in U$, and $v \in V$. Similarly the joint fuzzy relation $R_{B,A} \subset V \times U$, the transposition of $R_{A,B}$, is defined by the membership function $\mu_{R_{B,A}}(v,u)$ satisfying

$$\mu_{B,A}(v,u) = \mu_{R_{A,B}^T}(v,u) = \mu_B(v) \wedge u_{R_{B \rightarrow A}}(u/v). \tag{11}$$

Marginal Membership Functions --- The marginal membership functions, analogous to the Zadehian definition of the domain and range of R may be defined through the following operations:

$$\mu_A(u) = \underset{v}{V} \mu_{R_{A,B}}(u,v) \tag{12a}$$

$$= \underset{v}{V} \mu_{R_{B,A}^T}(u,v), \tag{12b}$$

$$\mu_B(v) = \underset{u}{V} \mu_{R_{A,B}}(v,u) \tag{13a}$$

$$= \underset{u}{V} \mu_{R_{B,A}^T}(v,u). \tag{13b}$$

Using the definition of transposition, and substituting (11) and (10) into (12b) and (13b) respectively, the marginal membership functions $\mu_A(u)$ and $\mu_B(v)$ can then be expressed in the alternative forms as shown in the following:

$$\mu_A(u) = \underset{v}{V} \mu_{R_{B,A}^T}(u,v)$$

$$= \underset{v}{V} \mu_{R_{B,A}}(v,u)$$

280

$$-\bigvee_v [\mu_B(v) \wedge \mu_{R_{B\to A}}(u/v)] \qquad (14)$$

$$\mu_B(v) = \bigvee_u [\mu_A(u) \wedge \mu_{R_{A\to B}}(v/u)]. \qquad (15)$$

Note that a fuzzy subset B will be induced from a fuzzy subset A and the conditional fuzzy relation $R_{A\to B}$. On the other hand, a subset A will be induced from a fuzzy subset B and the conditional relation $R_{B\to A}$. Therefore, both (14) and (15) are the same as the max-min operation as being used for the fuzzy compositional rule of inference by Zadeh [8].

Furthermore, substituting (10) and (11) into (12a) and (13a) respectively, $\mu_A(u)$ and $\mu_B(v)$ can be alternately given in the form

$$\mu_A(u) = \bigvee_v [\mu_A(u) \wedge \mu_{R_{A\to B}}(v/u)] \qquad (16)$$

$$\mu_B(v) = \bigvee_u [\mu_B(v) \wedge \mu_{R_{B\to A}}(u/v)]. \qquad (17)$$

From (12a), (13a), (16), and (17), the following two inequalities can be obtained readily.

$$\{\bigvee_v \mu_{R_{A\to B}}(v/u)\} \geqslant \mu_A(u) = \bigvee_v \mu_{R_{A,B}}(u,v), \qquad (18)$$

$$\{\bigvee_u \mu_{B\to A}(u/v)\} \geqslant \mu_B(v) = \bigvee_u \mu_{R_{B,A}}(u,v). \qquad (19)$$

Some important characteristics of fuzzy relations can be established. From (10) and (11), we have

$$\mu_{R_{A,B}}(u,v) \leqslant \mu_{R_{A\to B}}(v/u) \qquad (20)$$

$$\mu_{R_{B,A}}(v,u) \leqslant \mu_{R_{B\to A}}(u/v). \qquad (21)$$

as it should be. On the other hand, from (18) and (19), we have

$$\mu_A(u) \geqslant \mu_{R_{A,B}}(u,v) \quad ;\forall\, v \in V \qquad (22)$$

$$\mu_B(v) \geqslant \mu_{R_{B,A}}(v,u). \quad ;\forall\, u \in U \qquad (23)$$

Note that a given raw in $\mu_{R_{A,B}}(u,v)$ consists of entries which are less than or equal to $\mu_A(u)$; a given column in $\mu_{R_{B,A}}(v,u)$ consists of entries which are less than or equal to $\mu_B(v)$. These inequality relations between joint and conditional functions agree with those between joint and conditional probabilities in probability theory.

There are a number of ways that the conditional membership function can be constructed. We intend to summarize here the most frequently encountered types of implications in fuzzy logic with the objective of demonstrating their potential usefulness in applications.

$$(\text{Type 1}) \quad \mu_R(v/u) = \mu_A(u) \wedge \mu_B(v) \qquad (24)$$

$$(\text{Type 2}) \quad \mu_R(v/u) = (\mu_A(u) \wedge \mu_B(v)) \vee (1 - \mu_A(u)) \qquad (25)$$

$$(\text{Type 3}) \quad \mu_R(v/u) = 1 \wedge (1 - \mu_A(u) + \mu_B(v)) \qquad (26)$$

$$(\text{Type 4}) \quad \mu_R(v/u) = (1 - \mu_A(u)) \vee \mu_B(v) \qquad (27)$$

$$(\text{Type 5}) \quad \mu_R(v/u) = \begin{cases} 1 & \text{if } \mu_A(u) \leqslant \mu_B(v) \\ \mu_B(v) & \text{if } \mu_A(u) > \mu_B(u) \end{cases} \qquad (28)$$

The essential characteristics of each type of implication are summarized in Table 2.1. Types 1 and 5 share most of essential characteristics; Type 2, 3, and 4 share the similar characteristics mutually but are distinctly different from the other cluster of Type 1 and 5. The joint relation obtained from the Type 5 conditional relation is the exactly same as that of Type 1. The joint relations obtained from the Type 2, 3, and 4 conditional relations are not exactly the same but very similar to that of Type 1.

The analytical development made in this paper does confirm the existence of the mathematical relations between joint and conditional fuzzy relations. It also support the rational and the utilitarian value of the fuzzy compositional rule of inference as originally proposed by Zadeh [8]. If antecedents, $A's$, the consequences, $B's$, are noninteractive, then Type 1 implication, in particular, can be advantageously used.

Fuzzy Relations and Some Inverse Operations

Let us begin with some novel definitions which will be proved to be useful later on.

Bounded Fuzzy Subset --- Let U be the universe of discourse; let A, A_U, and A_L be fuzzy subsets of U. Then, a *bounded fuzzy subset*, $\bar{A}$, of U is a collection of fuzzy subsets

$$\bar{A} = \{A \mid \forall A \subset U \text{ and } A_L \subseteq A \subseteq A_U\} \qquad (29)$$

where A_U is called the "upper bound" of $\bar{A}$; A_L, the "lower bound" of $\bar{A}$.

The bounded fuzzy subset $\bar{A}$ can be alternatively characterized by the interval of the membership functions of A_U and A_L as follows:

$$\mu_{\bar{A}}(u) = [\mu_{A_L}(u), \mu_{A_U}(u)] \quad ;\forall\, u \in U. \qquad (30)$$

A fuzzy subset is considered as a special case of the bounded fuzzy subset where the upper bound and the lower bound are identical; that is, $A_U = A_L$. A bounded fuzzy subset is well illustrated by the hatched region as shown in Figure 1.

Sufficient Bounded Fuzzy Subset -- Let U be the universe of discourse. Assume that $\bar{A}$ and $\bar{X}$ represent bounded fuzzy subsets; X_U, X_L, and X, represent fuzzy subsets of U. Then, a *sufficient bounded fuzzy subset* of $\bar{A}$ is a bounded fuzzy subset

$$\bar{X} = \{X \mid \forall X \subset U, X_L \subseteq X \subseteq X_U, \text{ and } Inf(\bar{A}) \subseteq X_L, X_U \subseteq Sup(\bar{A})\}$$

$$(31)$$

where X_U and X_L are called an "upper sufficient bound" and a "lower sufficient found" of $\bar{A}$, respectively.

Necessary Bounded Fuzzy Subset --- Let U be the universe of discourse. Let $\bar{A}$ and $\bar{Y}$ be bounded fuzzy subsets; Y, Y_U, and Y_L are fuzzy subsets of U. Then a *necessary bounded fuzzy subset of $\bar{A}$* is a

	Type 1	Type 2	Type 3	Type 4	Type 5
$R_{A \to B}$	$A \times B$	$(A \times B) U (\neg A \times U)$	$\neg A \oplus B$	$\neg A U B$	$A \underset{g}{=} B$
$R_{A,B}$	$A \times B$	$(A \times B) U (\neg A \times A)$	$A \times (1 - A + B)$	$(A \times B) U (A \times \neg A)$	$A \times B$
$R_{A \to B} = R_{B \to A}^T$	Yes	No	No	No	No
$R_{A,B} = R_{B,A}^T$	Yes	No	No	No	Yes
$R_{A,B} = R_{A \to B}$	Yes	No	No	No	No
* $dom(R_{A,B}) = A$	Yes	Yes	Yes	Yes	Yes
* $dom(R_{B,A}^T) = A$	Yes	No	No	No	No
** $ran(R_{B,A}) = B$	Yes	Yes	Yes	Yes	Yes
** $ran(R_{A,B}^T) = B$	Yes	No	No	No	No
** $AoR_{A \to B} = B$	Yes	No	No	No	Yes

*: If $\mu_A(u) \leqslant \underset{v}{V} \mu_B(v)$, **: if $\mu_B(v) \leqslant \underset{u}{V} \mu_A(u)$

Table I Summary of Types of Implication

bounded fuzzy subset

$$\check{Y} = \{Y \mid {}^{\vee} Y \subset U, Y_L \subseteq Y \subseteq Y_U, \text{ and } Y_L \subseteq Inf(\check{A}), Sup(A) \subseteq Y_U\} \tag{32}$$

where Y_U and Y_L are called an "upper necessary bound" and a "lower necessary bound" of $\check{A}$ respectively.

The sufficient bounds and necessary bounds are also illustrated in Figure 1.

Greatest Lower Bound and Least Upper Bound --- Let $\hat{A}$ and $\check{B}$ be bounded fuzzy subsets in the universe of discourse U. Suppose A_U and B_U are upper bounds of $\check{A}$ and $\check{B}$, respectively; A_L and B_L, lower bounds of $\check{A}$ and $\check{B}$, respectively. A *greatest lower bound*, Lmax, and a *least upper bound*, Umin, are defined respectively as

$$Lmax = \int_U \mu_{A_L}(u) \ V \ \mu_{B_L}(u)/u, \tag{33}$$

$$Umin = \int_U \mu_{A_U}(u) \ \Lambda \ \mu_{B_U}(u)/u, \tag{34}$$

where $u \in U$. This relation is illustrated in Figure 2.

Ω*-Composition* --- Let A and B be fuzzy subsets on U, and $u \in U$. An Ω*-composition* of A and B, denoted by $A \ \Omega \ B$ is a binary operation, and is uniquely defined as

$$\mu_{A \Omega B}(u) = \mu_A(u) \ \omega \ \mu_B(u) \tag{35}$$

where

$$\mu_A(u) \ \omega \ \mu_B(u) = \begin{cases} [\mu_B(u), \ 1], & if \ \mu_A(u) \geqslant \mu_B(u) \\ [0, \mu_A(u)], & if \ \mu_A(u) < \mu_B(u). \end{cases}$$

Note that this Ω-composition is not commutative and is a different operation from the ω-composition proposed by Tsukamoto et al. [6].

$\hat{\Omega}$*-composition* --- Let A and B be fuzzy subsets of U, and $u \in U$. An $\hat{\Omega}$-composition of A and B, denoted by $A \ \hat{\Omega} \ B$, is defined as

$$\mu_{A \hat{\Omega} B}(u) = \mu_A(u) \ \hat{\omega} \ \mu_B(u). \tag{36}$$

where

$$\mu_A(u) \ \hat{\omega} \ \mu_B(u) = \begin{cases} \mu_B(u), & if \ \mu_A(u) \geqslant \mu_B(u) \\ 0, & if \ \mu_A(u) < \mu_B(u). \end{cases}$$

$\check{\Omega}$*-composition* --- Let A and B be fuzzy subsets of U, and $u \in U$. An $\check{\Omega}$*-composition* of A and B, denoted by $A \ \check{\Omega} \ B$ is defined as

$$\mu_{A \check{\Omega} B}(u) = \mu_A(u) \ \check{\omega} \ \mu_B(u), \tag{37}$$

where

$$\mu_A(u) \ \check{\omega} \ \mu_B(u) = \begin{cases} 1, & if \ \mu_A(u) \geqslant \mu_B(u) \\ \mu_A(u), & if \ \mu_A(u) < \mu_B(u). \end{cases}$$
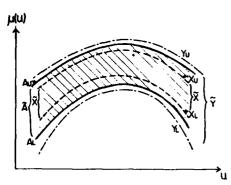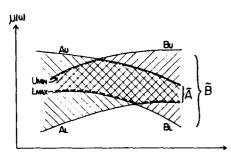
282

Figure 1   A bounded fuzzy subset.

Figure 2   Greatest Lower and Least Upper Bounds.

Note that both $\hat{\Omega}$-composition and $\check{\Omega}$-composition are the special cases of Ω-composition; the upper bound of Ω-composition is given by $\check{\Omega}$-composition; the lower bound of Ω-composition, by $\hat{\Omega}$-composition. The relation can be expressed as

$$Sup(A \; \Omega \; B) = A \; \overset{v}{\Omega} \; B \qquad (38)$$

$$Inf(A \; \Omega \; B) = A \; \hat{\Omega} \; B \qquad (39)$$

o-*fuzzy Inference* --- Let R be a fuzzy relation on $U \; x \; V$, and A be a fuzzy subset of U. For a given fuzzy subset B of V, we define $A = B \circ R^{-1}$, *o-fuzzy inference* or Zadehian fuzzy inference, by

$$\mu_A(u) = \underset{v}{V} \; [\mu_B(v) \; \Lambda \; \mu_{R^{-1}}(v,u)]. \qquad (40)$$

$\hat{\omega}$-*fuzzy Inference* --- Let $R \subset U \; x \; V$ be a fuzzy relation. For given fuzzy subset B of V, we define $A = B \; \hat{\omega} \; R$, the $\hat{\omega}$-*fuzzy infer-ence* by

$$\mu_A(u) = \underset{v}{V} \; [\mu_B(v) \; \hat{\omega} \; \mu_{R^{-1}}(v,u)], \qquad (41)$$

where $u \; \epsilon \; U$ and $v \; \epsilon \; V$.

$\check{\omega}$-*fuzzy Inference* --- Let $R \subset U \; x \; V$ be a fuzzy relation. For a given fuzzy subset B or V, we define $A = B \; \overset{v}{\omega} \; R$, the $\overset{v}{\omega}$-*fuzzy infer-ence*, by

$$\mu_a(u) = \underset{v}{\Lambda} \; [\mu_B(v) \; \overset{v}{\omega} \; \mu_{R^{-1}}(v,u)], \qquad (42)$$

where $u \; \epsilon \; U$ and $v \; \epsilon \; V$.

ω-*fuzzy Inference* --- Let $R \subset U \; x \; V$ be a fuzzy relation. For a given fuzzy subset B of V, we define $A = B \; \omega \; R$, the ω-*fuzzy infer-ence*, by

$$\mu_A(u) = \underset{v}{\cap} \; [\mu_B(v) \; \omega \; \mu_{R^{-1}}(v,u)] \qquad (43)$$

where $\underset{v}{\cap}$ denotes the operation to find the interval between the least upper bound, $Umin(u)$, and the greatest lower bound, $Lmax(u)$, of $\mu_B(v) \; \omega \; \mu_{R^{-1}}(v,u)$.

An alternative definition, making the use of $\hat{\omega}$-operation and $\overset{v}{\omega}$-operation, is presented as follows:

$$\mu_A(u) = \underset{v}{\cap} \; \mu_B(v) \; \omega \; \mu_{R^{-1}}(v,u)$$

$$= [Lmax(u), \; Umin(u)] \qquad (44)$$

where

$$Lmax(u) = \underset{v}{V} \; [\mu_B(v) \; \hat{\omega} \; \mu_{R^{-1}}(v,u)]$$

$$Umin(u) = \underset{v}{\Lambda} \; [\mu_B(v) \; \overset{v}{\omega} \; \mu_{R^{-1}}(v,u)].$$

The relationship between the different types of fuzzy inference described above can be best summed up through the following equa-tions in a concise manner

$$Sup(B \; \omega \; R^{-1}) = B \; \overset{v}{\omega} \; R^{-1}, \qquad (45)$$

$$Inf(B \; \omega \; R^{-1}) = B \; \hat{\omega} \; R^{-1}. \qquad (46)$$

Let us now point out some useful properties of the $\overset{v}{\omega}$- and $\hat{\omega}$-operation and their interplay with fundamental operators Λ and V. These identities will allow use to derive some theorems in the next sec-tion.

($\hat{\omega}$-operation) With $p,q,t \; \epsilon \; [0,1]$, $\hat{\omega}$-operation is defined as

$$p \; \hat{\omega} \; q = \begin{cases} q, & \text{if } p \geqslant q \\ 0, & \text{if } p < q. \end{cases} \qquad (47)$$

$$o \; p \; \hat{\omega} \; q \geqslant t \; \hat{\omega} \; q, \quad \text{if } p \geqslant t. \qquad (48)$$

$$o \; p \; \hat{\omega} \; q \leqslant p \; \Lambda \; q \qquad (49)$$

$$o \; p \; \Lambda \; (p \; \hat{\omega} \; q) = p \; \hat{\omega} \; q \qquad (50)$$

283

$$o \ (p \ \hat{\omega} \ q) \wedge q - p \ \hat{\omega} \ q \qquad (51)$$

$$o \ (p \vee t) \ \hat{\omega} \ q \leqslant p \ \hat{\omega} \ q \qquad (or \geqslant t \ \hat{\omega} \ q) \qquad (52)$$

$$o \ p \ \hat{\omega} \ (p \wedge q) - p \wedge q \leqslant q \ (or \leqslant p) \qquad (53)$$

$$o \ (p \wedge q) \ \hat{\omega} \ q - p \ \hat{\omega} \ q \qquad (54)$$

$$o \ (p \ \hat{\omega} \ q) \ \hat{\omega} \ q - p \ \hat{\omega} \ q \qquad (55)$$

$$o \ (p \ \hat{\omega} \ t) \wedge q \leqslant t \wedge q \quad ; ^\vee t. \qquad (56)$$

($\overset{v}{\omega}$-operation) With $p,q,t \in [0,1]$, $\overset{v}{\omega}$-operation is defined as

$$p \ \overset{v}{\omega} \ q - \begin{cases} 1, & if \ p \geqslant q \\ p, & if \ p < q. \end{cases} \qquad (57)$$

$$o \ p \wedge (p \ \overset{v}{\omega} \ q) - p \qquad (58)$$

$$o \ (p \ \overset{v}{\omega} \ q) \wedge q - p \wedge q \qquad (59)$$

$$o \ p \ \overset{v}{\omega} \ (q \vee t) \leqslant p \ \overset{v}{\omega} \ q \quad (or \leqslant p \ \overset{v}{\omega} \ t) \qquad (60)$$

$$o \ (p \vee t) \ \overset{v}{\omega} \ q \geqslant p \ \overset{v}{\omega} \ q \quad (or \geqslant t \ \overset{v}{\omega} q) \qquad (61)$$

$$o \ p \ \overset{v}{\omega} \ (p \wedge q) - 1 \qquad (62)$$

$$o \ (p \wedge q) \ \overset{v}{\omega} \ q - p \ \overset{v}{\omega} \ q \qquad (63)$$

$$o \ (p \ \overset{v}{\omega} \ q) \ \overset{v}{\omega} \ q - p \ \overset{v}{\omega} \ q \qquad (64)$$

$$o \ (p \ \overset{v}{\omega} \ t) \wedge q \geqslant p \wedge q \quad ; ^\vee t. \qquad (65)$$

In this section, some essential theoretical results will be presented in the form of eleven theorems. The proofs will be found in [9].

For the further investigation, let $\bar{A}$ and $\bar{B}$ be bounded fuzzy subsets, and let A and B be fuzzy subsets of $\bar{A}$ and $\bar{B}$, respectively.

Theorem 1 For a given fuzzy subset $A \subset U$ and a fuzzy relation $R \subset U \ x \ V$, we have

$$A \cap dom(R) \subseteq (A \ o \ R) \ o \ R^{-1} \qquad (66)$$

Theorem 2 For a given fuzzy subset $A \subset U$ and a fuzzy relation $R \subset U \ x \ V$, we have

$$A \ \hat{\Omega} \ dom(R) \subseteq (A \ o \ R) \ \hat{\omega} \ R^{-1} \qquad (67)$$

Theorem 3 For a given fuzzy subset A of U and a fuzzy relation R in UxV we have

$$A \ \overset{v}{\omega} \ R \subseteq (A \ o \ R) \ \overset{v}{\omega} \ R^{-1} \qquad (68)$$

Theorem 4 For a given fuzzy subset $B \subset V$ and a fuzzy relation $R \subset U \ x \ V$, we have

$$B \cap ran(r) \subseteq (B \ o \ R^{-1}) \ o \ R. \qquad (69)$$

Theorem 5 For a given fuzzy subset $B \subset V$ and a fuzzy relation $R \subset U \ x \ V$, we have

$$B \ \hat{\Omega} \ ran(R) \subseteq (B \ \hat{\omega} \ R^{-1}) \ o \ R. \qquad (70)$$

Theorem 6 For a given fuzzy subset $B \subset V$ and a fuzzy relation $R \subset U \ x \ V$, we have

$$B \cap ran(r) \supseteq (B \ \overset{v}{\omega} \ R^{-1}) \ o \ R. \qquad (71)$$

Theorem 7 For a given fuzzy subset $B \subset V$ and a fuzzy relation $R \subset U \ x \ V$, we have

$$(B \ \overset{v}{\omega} \ R^{-1}) \ o \ R \subseteq (B \ o \ R^{-1}) \ o \ R. \qquad (72)$$

Theorem 8 For a given fuzzy subset B of V and a fuzzy relation R in $U \ x \ V$, we have

$$(B \ \hat{\omega} \ R^{-1}) \ o \ R \subseteq (B \ o \ R^{-1}) \ o \ R. \qquad (73)$$

This is obvious from the definition of $\hat{\omega}$- and o-operations.

From definitions, we have some important properties;

$$(A \ \hat{\Omega} \ dom(R)) \subseteq (A \cap dom(R)) \subseteq A, \qquad (74)$$

$$A \subseteq A \ \overset{v}{\omega} \ R \subseteq (A \ \overset{v}{\Omega} \ sdom(R)). \qquad (75)$$

Theorem 9 For a given fuzzy subset $A \subset U$ and a fuzzy relation $R \subset U \ x \ V$, we have

$$A \ o \ R - (A \cap dom(R)) \ o \ R. \qquad (76)$$

Note that Theorem 9 indicates that $\{A \cap dom(R)\}$ and A produce the same results through o-inference. In words, theorem 9 indicates that $\{A \cap dom(R)\}$ is a necessary and a sufficient lower bound of A; that is, $\{A \cap dom(R)\}$ is the lower bound of A.

Theorem 10 For a given fuzzy subset A of U and a fuzzy relation R on $U \times V$, we have

$$A \circ R \subseteq (A \overset{\vee}{\omega} R) \circ R. \tag{77}$$

Theorem 11 For a given fuzzy subset $B \subset V$ and a fuzzy relation $R \subset U \times V$, we have

$$(B \cap ran(R)) \circ R^{-1} = B \circ R^{-1}. \tag{78}$$

Thus $B \cap ran(R)$ is a sufficient lower bound of B.

Conclusion

Topics associated with the fuzzy relational function, which has great application in solving engineering problems, have been investigated in the course of this work.

The concept of conditional, joint, and marginal fuzzy relations has been introduced, extended, and put into right perspective. Now, the existence of their mutual relationship can be explained in the more satisfactory manner. In addition, these mutual relationship can be established through explicit mathematical expression in a consistent and more unified manner. This study has revealed some intrinsic properties of the Zadehian inference from a different perspective. The rationale of this inference, hence, has picked up stronger supportive evidence.

Furthermore, the precise condition for a fuzzy subset A to satisfy the relation $A \circ R_{A \rightarrow B} = B$ (*modus ponens*) has been established.

A clear meaning of so-called "interactiveness" - a concept which may be analogous to the "dependence" in probability theory - seems desirable at this point; however, this work would be for further inquiry.

In the last second half of this paper, a novel method to search the upper and lower bounds for a solution to the fuzzy inverse problem has been proposed. It has been proven that the proposed method, i.e., Ω-composition, is more flexible than the technique currently available. The major advantage of the proposed method lies in the fact that it can establish the so-called "sufficient bound" and "necessary bound" as well. Although various methods for establishing bounds for inverse problem does exist, this study has provided a quite powerful approach to search bounds.

REFERENCES

[1] Sembi, B. S. and Mamdani, E. H., 1979. "On the Nature of Implication in Fuzzy Logic," *Proc. 9th Int. Symp. on Multiple-Valued Logic* (Bath, England), pp. 143-151.

[2] Mizumoto, M., Fukami, S., and Tanaka, K., 1979. "Several Methods for Fuzzy Conditional Inferences," *Proc. IEEE Conf. on Decision & Control.* (Florida, December 12-14) pp. 777-782.

[3] Hisdal, E., 1978. "Conditional Probability Independence and Noninteractions," *Fuzzy Sets and Systems.* 1: 283-397.

[4] Sanchez, E., (1976).

[5] Tashiro, T., Terano, R., and Tsukamoto, Y., (1978). "Inverse of Fuzzy Correspondence and Evolutionary Diagnosis," *Proc. Int. Conf. on Cybernetics and Society,* Vol. 2, pp. 938-941, Tokyo.

[6] Tsukamoto, Y. and Terano, T., (1977). "Failure Diagnosis by Using Fuzzy Logic," *Proc. IEEE Conf. on Decision and Control,* pp. 1390-1394, New Orleans, Lu.

[7] Tsukamoto, Y., (1979). *Fuzzy Logic Based on Lukasiewicz Logic and Its Applications to Diagnosis and Control,* Ph.D. Dissertation, Tokyo Institute of Technology.

[8] Zaden, L. A., (1973). "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes," *IEEE Trans. on Sys., Man, and Cybern.,* Vol. SMC-3, No. 1, pp. 28-45.

[9] Togai, Masaki, (1982). *Principles and Applications of Fuzzy Inference: A New Approach to Decision-Making Processes in Ill-Defined System.* Ph.D. Dissertation, Duke University.

REGULAR TERNARY LOGIC FUNCTIONS
--- TERNARY LOGIC FUNCTIONS SUITABLE FOR TREATING AMBIGUITY ---

Masao Mukaidono

Faculty of Engineering, Meiji University
1-1-1 Higashi-mita, Tama-ku, Kawasaki-shi, JAPAN 214

Abstract

A special group of ternary functions, called regular ternary logic functions, are defined. These functions are useful in switching theory, programming languages, algorithm theory and many other fields, if we are concerned with the indefinite state in such fields. This paper describes the fundamental properties and representations of the regular ternary logic functions.

1. Introduction

Logics and algorithms are generally based on the two-valued principle, that is, true or false, or yes or no. However, in some cases, we experience a state in which it is impossible or unnecessary to decide true or false. For example, each value of a signal in a logic circuit, which takes 0 or 1 in a steady state, changes from 0 to 1 or from 1 to 0 in a transient state; that is, it is impossible to decide whether the value is 0 or 1. The initial states of sequential circuits is another example, where it is difficult to know whether the value is 0 or 1 in many cases. Furthermore, it may be said that an algorithm does not stop for a given data, or that some data are not applicable to the algorithem. In the cases mentioned above, we may use ternary logic (three-valued logic), instead of binary logic (two-valued logic), in which the third truth value is introduced to represent an ambiguous state apart from true and false.

On the other hand, ternary functions have been studied for some time from the standpoint of their functional completeness or representation. When applying ternary functions to various fields of engineering, we seldom use all the ternary functions; instead, we employ only some subsets, which have special properties or meanings. In fact, Mukaidono [1980] has introduced some special subsets of ternary functions, called regular, normal and uniform, respectively, which have important and useful properties.

The present paper discusses in detail a special group of ternary functions, called regular ternary logic functions and introduced firstly in Mukaidono [1980], which are significant if the third truth value is considered to represent an ambiguous state. That is, regular ternary logic functions, which will be studied in this paper, are suitable for treating ambiguity. In Section 2, we shall introduce regular ternary logic functions from three different standpoints and show that they are all the same definitions. A representation of regular ternary logic functions is discussed in Section 3, and their axioms and functional completeness are explained in Section 4. Finally, in Section 5, the canonical form, which is determined uniquely for any given regular ternary logic function, is studied.

2. Regular Ternary Logic Functions

A ternary function is defined as follows, using the symbol 1/2 as the third truth value in contrast to 0 (false) and 1 (true): Letting $V=\{0, 1/2, 1\}$, a n-variable ternary function F is defined to be a mapping from V^n to V:

$$F: V^n \rightarrow V.$$

Here, we will interpret the truth value 1/2 as " uncertain 0 or 1", that is, " ambiguous". Then, we can define the truth tables of the logic connectives AND($\cdot$), OR($+$) and NOT($^-$) as in Table 1. Also, let us consider the ternary functions defined by the following condition:

(C1) the ternary functions which can be represented by well-formed logic formulas consisting of variables $x_1, \cdots, x_n$, constants 0, 1/2, 1

A·B	0	1/2	1
0	0	0	0
1/2	0	1/2	1/2
1	0	1/2	1

AND: A·B

A+B	0	1/2	1
0	0	1/2	1
1/2	1/2	1/2	1
1	1	1	1

OR: A+B

A	0	1/2	1
	1	1/2	0

NOT: $\overline{A}$

Table 1: Truth tables of ternary AND, OR and NOT.

and logic connectives AND($\cdot$), OR($+$) and NOT
($\overline{}$) defined in Table 1. Hereafter, we call a
ternary function satisfying the above condi-
tion C1 a <u>ternary function representable by a</u>
<u>logic formula</u>.

[Note 1] The truth tables of Table 1 are called
Kleene's ternary logic system (Kleene[1952]).
The same truth tables as Table 1 were used in-
dependently by Goto[1949] to analyze indefinite
behaviors of relay circuits.

Here, let us define a partial ordered relation
"$<<$" concerning ambiguity on $V=\{0,1/2,1\}$ and V^n as
follows:

[Definition 1] $0<<1/2$, $1<<1/2$, $i<<i$, $i\epsilon V$.

In the relation $<<$, 0 and 1 are not comparable
with each other. The relation can be extended among
V^n as follows: For two elements $A=(a_1,\cdots,a_n)$ and
$A'=(a_1',\cdots,a_n')$ of V^n, $A'<<A$ if and only if $a_i'<<a_i$
for all values of i. If $A'<<A$, then A' is said to
be less ambiguous than or equal to A.

[Example 1] Suppose $A_1=(0,1/2,1/2)$, $A_2=(1,1/2,0)$
and $A_3=(1/2,1/2,1/2)$; then $A_1<<A_3$, $A_2<<A_3$, where A_1
and A_2 are not comparable with each other.

As a condition for a ternary function F to be
significant when the truth value 1/2 is assumed to
represent an ambiguous state, it will be postulated
that if the value of F(A) is definite, that is, 0 or
1, then, F(A') takes an equal value for every ele-
ment A' which is less ambiguous than or equal to A;
that is,

(C2) Regularity: if $F(A)\epsilon B=\{0,1\}$; then, $F(A')=F(A)$
for every A' such as $A'<<A$.

[Definition 2] A ternary function F is called a
<u>regular ternary logic function</u> if and only if F
satisfies the regularity condition C2.

[Example 2] Let the two-variable ternary functions
F_1 and F_2 be given by Table 2. Then, F_1 is a regu-
lar ternary logic function while F_2 is not. In fact,
$(1,1)<<(1/2,1)$, but $F_2(1,1)=1\neq0=F_2(1/2,1)$.

[Note 2] The condition of regularity C2 defined
above is an extension of Kleene's definition to

x_2 \ x_1	0	1/2	1
0	1	1/2	0
1/2	1	1/2	1/2
1	1	1/2	1/2

x_2 \ x_1	0	1/2	1
0	1/2	1/2	1
1/2	1/2	1/2	1
1	0	0	1

F_1-- Regular ternary F_2-- Non-regular ternary
logic function   logic function

Table 2: Example 2.

n-variable ternary functions, where Kleene's
original definition (Kleene[1952]) of regularity
for a truth table is as follows: The truth table
never takes 0 or 1 as an entry in the "1/2 row
(or column)" unless this entry 0 or 1 occurs uni-
formaly throughout its entire column (or row,
respectively).

Next, a ternary function which satisfies the
condition,

(C3) Monotonocity for ambiguity: if $A'<<A$, then
$F(A')<<F(A)$,
is called an <u>A-ternary logic function</u>. It is known
(Mukaidono[1978b]) that A-ternary logic functions
can be applied to design fail-safe logic circuits by
letting 1/2 correspond to a failure state.

[Note 3] A ternary function F which satisfies the
condition C3 and, also, the condition of normal-
ity (Mukaidono[1980]), that is, if $A\epsilon B^n=\{0,1\}^n$,
then $F(A)\epsilon B$, is called a B-ternary logic function
(Mukaidono[1972]) and is applied to detecting
hazards (Yoeli and Rinon[1964], Mukaidono[1978])
and fail-safe logic (Mukaidono[1969]).

Thus far, three different conditions C1, C2 and
C3 have been defined for ternary functions. In the
following, we will prove that these three conditions
are equivalent to each other.

[Theorem 1] F is a regular ternary logic function
if and only if F is a A-ternary logic function.
(Proof) Let us suppose that if $F(A)\epsilon B$, then $F(A)=$
$F(A')$ for every A' such as $A'<<A$. If $F(A)=1/2$, then
it is evident that $F(A')<<F(A)=1/2$ holds for every
A'. If $F(A)\epsilon B$, then $F(A')<<F(A)$ holds for every A'
such as $A'<<A$ by the supposition. That is, it is
always valid that if $A'<<A$, then $F(A')<<F(A)$. Con-
versely, let us suppose that if $A'<<A$, then $F(A')<<$
$F(A)$. If $F(A)\epsilon B$, then $F(A')<<F(A)$ implies $F(A')\epsilon B$.
(Q.E.D.)

[Theorem 2] If F is a ternary function representable
by a logic formula, then F is a regular ternary logic
function.
(Proof) It will be shown by induction concerning
the number of logic connectives. It is evident that
the constants 0, 1/2 and 1, and each variable $x_1,\cdots$,
x_n satisfy the condition C2. Suppose that all ter-
nary functions representable by logic formulas, in
which the number of logic connectives is smaller
than or equal to n, satisfy the condition C2. Next,
let us suppose that F is a ternary function repre-
sentable by a logic formula in which the number of
logic connectives is n+1. Hereafter, for simplicity,
we will identify a logic formula with the ternary
function represented by the formula. F is one of
$\overline{F}_1$, $F_1 \cdot F_2$ and F_1+F_2. $\overline{F}_1$ satisfies the condition C2
because of the fact that $F_1(A')<<F_1(A)$ is equal to
$\overline{F}_1(A')<<\overline{F}_1(A)$. Suppose that $A'<<A$ and $(F_1 \cdot F_2)(A')$
$\not<(F_1 \cdot F_2)(A)$. Then, this fact leads to one of
(1) $(F_1 \cdot F_2)(A)=0$ and $(F_1 \cdot F_2)(A')\neq0$, (2) $(F_1 \cdot F_2)(A)=$
1 and $(F_1 \cdot F_2)(A')\neq1$. Either case does not hold as
shown below. If $(F_1 \cdot F_2)(A)=0$, then $F_1(A)=0$ or $F_2(A)$

287

=0. By the assumption of deduction, we can obtain that $F_1(A')=0$ or $F_2(A')=0$, that is $(F_1 \cdot F_2)(A')=0$. This contradicts the assumption. It is similar in the case of (2). Therefore, $F_1 \cdot F_2$ satisfies the condition C2. Next, suppose that $A' << A$ and $(F_1+F_2)(A') \not\ll (F_1+F_2)(A)$. In a similar manner, we can show that F_1+F_2 satisfies the condition C2, because $(F_1+F_2)(A)=0$ and $(F_1+F_2)(A)=1$ lead to a contradiction. From the above, it has been shown that all ternary functions representable by logic formulas satisfy the condition C2. (Q.E.D.)

The converse of Theorem 2, that is, every regular ternary logic function can be represented by a logic formula, will be shown in the next section.

3. Reprentation of Regular Ternary Logic Function

A <u>literal</u> is a variable x_i or $\bar{x}_i$, the negation of x_i. A conjunction of one or more literals is called a <u>simple phrase</u> if it does not contain a literal and its negation, $x_i \cdot \bar{x}_i$, simultaneously for at least one variable x_i, and is called a <u>complementary phrase</u> otherwise. A disjunction of one or more literals is called a <u>simple clause</u> if it does not contain a literal and its negation, $x_i + \bar{x}_i$, simultaneously for at least one variable x_i, and is called a <u>complementary clause</u> otherwise. In the above definitions, it is assumed that any repeated literals are removed.

[Note 4] As evident from Table 1, $x \cdot \bar{x} = 0$ and $x + \bar{x} = 1$ when $x=1/2$ do not hold in Kleene's system. Therefore, we can not ignore conjunctions and disjunctions containing a literal and its negation simultaneously.

[Definition 3] Let $A=(a_1, \cdots, a_n)$ be an element of V^n. Then, A and a simple phrase $\alpha = x_1^{a_1} \cdots \cdots x_n^{a_n}$ (simple clause $\beta = x_1^{a_1} + \cdots + x_n^{a_n}$) correspond to each other if the following conditions hold: If $a_i=0$, then $x_i^{a_i} = \bar{x}_i$ ($x_i^{a_i}=x_i$); if $a_i=1$, then $x_i^{a_i}=x_i$ ($x_i^{a_i} = \bar{x}_i$); and if $a_i=1/2$, then there is no variable x_i in $\alpha(\beta)$.

[Example 3] Let $A=(1,1/2,0)$. Then, the simple phrase α corresponding to A is $\alpha = x_1 \cdot \bar{x}_3$, and the simple clause β corresponding to A is $\beta = \bar{x}_1 + x_3$.

[Definition 4] Let $A=(a_1, \cdots, a_n)$ and $A'=(a_1', \cdots, a_n')$ be any two elements of V^n. Then, it is said that A and A' are disjoint to each other and written as $A \cap A' = \emptyset$ if there is i in $\{1, \cdots, n\}$ such that a_i is 0 or 1 and $a_i = \bar{a}_i'$.

[Lemma 1] Let A be any element of V^n and α, β be the corresponding simple phrase and simple clause, respectively. Then,
(1) $A' << A$ iff $\alpha(A')=1$,

(2) $A' \cap A = \emptyset$ iff $\alpha(A')=0$,
(3) $A' \not\ll A$ and $A' \cap A \neq \emptyset$ iff $\alpha(A')=1/2$,
(4) $A' << A$ iff $\beta(A')=0$,
(5) $A' \cap A = \emptyset$ iff $\beta(A')=1$,
(6) $A' \not\ll A$ and $A' \cap A \neq \emptyset$ iff $\beta(A')=1/2$.
(Proof) Let $A=(a_1, \cdots, a_n)$ and $\alpha = x_{i1}^{a_{i1}} \cdots \cdots x_{ik}^{a_{ik}}$, where $a_{ij}(j=1, \cdots, k)$ is 0 or 1 and other elements of A are 1/2. For an element $A'=(a_1', \cdots, a_n')$, $\alpha(A')=1$ if and only if the value of $x_{ij}^{a_{ij}}$ is 1 for all j's ($1 \leq j \leq k$). This means that if a_{ij} is 0 or 1, then $a_{ij}'=a_{ij}$, that is, $A' << A$. Therefore, (1) is justified. Similarly, $\alpha(A')=0$ if and only if there is at least one j such that $a_{ij}=\bar{a}_{ij}'$, that is, $A' \cap A = \emptyset$. Thus, we arrive at (2). Also, (3) is derived directly from (1) and (2). In a similar manner, we can show (4), (5) and (6). (Q.E.D.)
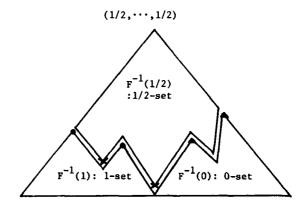
[Theorem 3] Let F be a regular ternary logic function and A be an element of V^n. Then,
(1) if $F(A)=1$, then $F(A')=1$ for every A' such that $A' << A$,
(2) if $F(A)=0$, then $F(A')=0$ for every A' such that $A' << A$,
(3) if $F(A)=1/2$, then $F(A')=1/2$ for every A' such that $A << A'$.
(Proof) These are evident from the condition of regularity (C2) and monotonicity for ambiguity (C3). (Q.E.D.)

Let F be an n-variable regular ternary logic function. Then, $F^{-1}(1)$, $F^{-1}(0)$ and $F^{-1}(1/2)$ represent the subsets of V^n mapped to 1, 0 and 1/2, and are called the 1-set, 0-set and 1/2-set, respectively. Theorem 3 indicates that $F^{-1}(1)$, $F^{-1}(0)$ and $F^{-1}(1/2)$ are partial ordered sets in regard to the relation << and that the sets $F^{-1}(1)$ and $F^{-1}(0)$ are determined uniquely by their maximal elements while $F^{-1}(1/2)$ is determined uniquely by its minimal elements (Figure 1). Here, of course, $F^{-1}(1) \cup F^{-1}(0) \cup F^{-1}(1/2) = V^n$ holds. In Figure 1, the symbol

Fifure 1: $V^n = F^{-1}(1) \cup F^{-1}(0) \cup F^{-1}(1/2)$.

● indicates the maximal elements of the 1-set, the symbol ▲ the maximal elements of the 0-set, and the symbol ✗ the minimal elements of the 1/2-set.

[Theorem 4] Any regular ternary logic function F can be represented by the logic formula

$$F = F^1 + (1/2) \cdot F^0,$$

where F^1 is the disjunction of simple phrases corresponding to all the maximal elements of the 1-set of F and where F^0 is the conjunction of simple clauses corresponding to all the maximal elements of the 0-set of F.

(Proof) Let A' be any element of V^n and F(A')=1. Then, there is a maximal element A in 1-set of F such that A'<<A. Hence, there is a simple phrase α corresponding to A in F^1, where α(A')=1 (Lemma 1(1)). Therefore, F^1(A')=1, that is, $(F^1+(1/2)\cdot F^0)$(A') =1. Next, suppose F(A')=1/2. Then, A' does not belong to either the 1-set or the 0-set of F. Hence, there is no simple phrase in F^1 and no simple clause in F^0 corresponding to A such that A'<<A. As a result, F^1(A')≠1 and F^0(A')≠0 (Lemma 1(1) and (4)). F^0(A')≠0 means that F^0(A')=1 or F^0(A')=1/2. Thus, we can show that $(F^1+(1/2)\cdot F^0)$(A')=1/2. Finally, suppose F(A')=0. Then, there is a simple clause corresponding to A such that A'<<A in F^0. Therefore, F^0(A')=0 holds (Lemma 1(4)). On the other hand, A'∩A=∅ is valid for every element A of the 1-set of F bacause A' belongs to the 0-set. That is, F^1(A')=0 is justified by Lemma 1(2). From the above, we have $(F^1+(1/2)\cdot F^0)$(A')=0.   (Q.E.D.)

    As we have seen, the three conditions C1, C2 and C3 described in the preceeding section are equivalent to each other. It is apparent from the above proof that F^1 and F^0 are determined uniquely (ignoring the order of phrases or clauses) for any given regular ternary logic function F. Therefore, the logic formula described in Theorem 4 can be used as a canonical form of regular ternary logic functions. In Section 5, we will consider another canonical form called the canonical disjunctive form.

[Example 4] Let us represent F_1 of Table 2 in Example 2 by a logic formula based on the above theorem. The set of maximal elements of the 1-set is {(0,1/2)} and that of the 0-set is {(1,0)}. Therefore, we have $F_1 = \overline{x}_1 + (1/2) \cdot (\overline{x}_1 + x_2)$.

4. Axioms and Functional Completeness of Regular
Ternary Logic Functions

    Any regular ternary logic function can be represented by a logic formula conposed of constants 0, 1/2 and 1, and logic connectives AND(·), OR(+) and NoT(⁻) defined by Table 1. As an algebraic system, the set of regular ternary logic functions safisfies the following equalities which also hold in Boolean algebra:
(1) the commutative laws A+B=B+A, A·B=B·A,
(2) the associative laws A+(B+C)=(A+B)+C,

A·(B·C)=(A·B)·C,
(3) the absorption laws A+(A·B)=A, A·(A+B)=A,
(4) the distributive laws A·(B+C)=(A·B)+(A·C),
                          A+(B·C)=(A+B)·(A+C),
(5) the idempotent laws A+A=A, A·A=A,
(6) De Morgan's laws $\overline{(A+B)}=\overline{A}\cdot\overline{B}$, $\overline{A\cdot B}=\overline{A}+\overline{B}$,
(7) the double negation law $\overline{\overline{A}}=A$,
(8) the least element 0+A=A, 0·A=0,
(9) the greatest element 1+A=1, 1·A=A,
(10) Kleene's laws $(A\cdot\overline{A})+B+\overline{B}=B+\overline{B}$, $A\cdot\overline{A}\cdot(B+\overline{B})=A\cdot\overline{A}$,
(11) center $1/2=\overline{1/2}$,
except
(the complementary laws) $A+\overline{A}=1$, $A\cdot\overline{A}=0$.

    The equalities (1)--(10) except (11) are equivalent to axioms of Kleene algebra or fuzzy algebra and have been studied in detail in Mukaidono[1981]. The element satisfying (11) is called a center. The regular ternary logic functions satisfy the axioms of Kleene (or fuzzy) algebra with a center.

    Next, we will examine these regular ternary logic functions from a standpoint of functional completeness. The set of logic connectives {0, 1/2, 1, +, ·, ⁻} (we consider constants as 0-variable logic connectives) cannot represent all ternary functions; that is, it is not functionally complete for ternary functions, but, as mentioned above, it is functionally complete in a strong sense (Mukaidono[1980]) for regular ternary logic functions. That is, any regular ternary logic function can be represented by {0,1/2,1,+,·,⁻} ; and conversely, a ternary function represented by {0,1/2,1,+,·,⁻} is always regular.

    Let us define ternary NOR(↑) and NAND(↓) as in Table 3.

[Theorem 5] The set of logic connectives {0,1/2,↑} is functionally complete for regular ternary logic functions.
(Proof) It is shown by $\overline{A}=A{\uparrow}A$, $1=\overline{0}$, $A+B=\overline{A{\uparrow}B}$ and $A\cdot B=\overline{A}{\uparrow}\overline{B}$. (Q.E.D.)

[Theorem 6] The set of logic connectives {0,1/2,↓} is functionally complete for regular ternary logic functions.
(Proof) It is shown by $\overline{A}=A{\downarrow}A$, $1=\overline{0}$, $A+B=\overline{A}{\downarrow}\overline{B}$ and $A\cdot B=\overline{A{\downarrow}B}$. (Q.E.D.)

[Note 5] The following problem arises: if a non-regular ternary logic function is added to the set of regular ternary logic functions, is the

A\B	0	1/2	1
0	1	1/2	0
1/2	1/2	1/2	0
1	0	0	0

NOR: A↑B

A\B	0	1/2	1
0	1	1	1
1/2	1	1/2	1/2
1	1	1/2	0

NAND: A↓B

Table 3: Truth tables of ternary NOR and NAND.

new set always functionally complete for ternary function ? That is, are regular ternary logic functions maximal ? The answer is negative. In fact, one-variable ternary functions $u_1, \cdots, u_6$ of Table 4 are non-regular, and even if one of them is added to the family of regular ternary logic functions, they are not functionally complete for ternary functions. But it can be proved that if any non-regular one-variable ternary function except those of Table 4 is added to the set of regular ternary logic functions, then they are functionally complete for ternary functions.

x	0	1/2	1
$u_1(x)$	0	0	1/2
$u_2(x)$	1/2	0	0
$u_3(x)$	1/2	0	1/2
$u_4(x)$	1/2	1	1/2
$u_5(x)$	1/2	1	1
$u_6(x)$	1	1	1/2

Table 4: Non-regular one-variable ternary functions.

5. Canonical Form of Regular Ternary Logic Functions

In this section, we shall introduce a canonical form for regular ternary logic functions, which is different from that of Theorem 4. We shall also discuss on the methods to obtain such a canonical form. Any logic formula representing a regular ternary logic function F can be expanded into a disjunctive form

$$F = \gamma_1 + \cdots + \gamma_m,$$

where $\gamma_i (i=1, \cdots, m)$ is a product term, because the distributive, absorption, De Morgan's, idempotent and other laws stand valid as stated in the preceding section. Here, each product term γ_i is one of the following three types:

type 1: ---------- α,
type 2': ------(1/2)$\cdot \alpha$,
type 3': -------- β,

where α is a simple phrase and β is a complementary phrase as described in Section 3. If a product term $(1/2) \cdot \beta$ (β is a complementary phrase) exists, then we can omit 1/2 and it is equal to type 3' because $x_i \cdot \overline{x}_i \leq 1/2$ stands always true. If a variable x_i does not exist in a product term $(1/2) \cdot \alpha$ of type 2', then the following relation holds:

$$(1/2) \cdot \alpha = (1/2) \cdot (x_i + \overline{x}_i) \cdot \alpha = (1/2) \cdot \alpha \cdot x_i + (1/2) \cdot \alpha \cdot \overline{x}_i,$$

as $x_i + \overline{x}_i \geq 1/2$ is always valid. In a similar manner, if a variable x_i does not exist in a complementary phrase β of type 3', then

$$\beta = \beta \cdot (x_i + \overline{x}_i) = \beta \cdot x_i + \beta \cdot \overline{x}_i$$

holds, since there is a factor $x_j \cdot \overline{x}_j$ in β for a variable x_j where $x_j \cdot \overline{x}_j \leq 1/2 \leq x_i + \overline{x}_i$ always holds. From the above, we can expand α of type 2' and β of type

3' into disjunctions of product terms in which all variables exist, respectively. A simple phrase and complementary phrase in which all variables exist are called a <u>minterm</u> and <u>complementary minterm</u>, respectively.

Consequently, any regular ternary logic function can always be expanded into the disjunction of the following three types of product terms:

type 1: $\alpha = x_{ij}^{a_{ij}} \cdots \cdots x_{ik}^{a_{ik}}$ ----- simple phrase,

type 2: $(1/2) \cdot \alpha' = (1/2) \cdot x_1^{a_1} \cdots \cdots x_n^{a_n}$ ---α' is a minterm

type 3: $\beta = x_1^{a_1} \cdots \cdots x_n^{a_n} \cdot x_{ij}^{1-a_{ij}} \cdots \cdots x_{ik}^{1-a_{ik}}$ ----- complementary minterm,

where a_i is 0 or 1.

Next, let us examine the relations of each type of product term. Here, for two product term γ and γ', if all literals of γ exist in γ' as well, then it is written as $\gamma \supseteq \gamma'$. In this case, $\gamma + \gamma' = \gamma$ is true; that is, γ' is absorbed by γ in accordance with the absorption law.

[Definition 5] Let $A = (a_1, \cdots, a_n)$ be an element of V^n. Then, the element A corresponds to a product term of type 2 or type 3 if the following relations holds:

if $a_i = 0$, then $x_i^{a_i} = x_i$,
if $a_i = 1$, then $x_i^{a_i} = \overline{x}_i$,
if $a_i = 1/2$, then $x_i^{a_i} = x_i \cdot \overline{x}_i$.

[Example 5] $(0,0,1)$ corresponds to a product term of type 2, $(1/2) \cdot \overline{x}_1 \cdot \overline{x}_2 \cdot x_3$, and $(0,1/2,1)$ corresponds to that of type 3, $\overline{x}_1 \cdot x_2 \cdot \overline{x}_2 \cdot x_3$. Product terms of type 2 correspond to elements of B^n, and product terms of type 3 to those of $V^n - B^n$, where $B^n = \{0,1\}^n$.

[Lemma 2] Let α be a product term of type 1, α' be that of type 2 or type 3, and A and A' be elements corresponding to α and α', respectively. Then,
(1) if $\alpha(A') = 1$, then $\alpha \supseteq \alpha'$,
(2) $\alpha'(A) = 1/2$ if and only if $A' << A$.
(Proof) It is shown by the definitions of type 1, type 2, type 3 and Definition 5. (Q.E.D.)

[Definition 6] If a regular ternary logic function F is represented by a logic formula

$$F = \gamma_1 + \cdots + \gamma_m,$$

then it is said that F is in the <u>canonical disjunctive</u> form, where $\gamma_i (i=1, \cdots, m)$ is one of type 1, type 2 or type 3 and $\gamma_i \not\supseteq \gamma_j$ for all i,j (i≠j).

[Theorem 7] Any regular ternary logic function can be represented uniquely (ignoring the order of the product terms) by the canonical disjunctive form.
(Proof) Let us suppose $F_1 = \gamma_1 + \cdots + \gamma_s$ and $F_2 \cdot \gamma_1' + \cdots + \gamma_t'$ are two different canonical disjunctive forms of a regular ternary logic function F. (It is evident from the above discussion that there is at least one canonical disjunctive form for F). Now, we can suppose that a product term γ exists in F_1 but not in F_2 without loss of generality. First, assume γ is a product term of type 1, that is, a simple phrase. If A is an element corresponding to

290

γ, then $F_1(A)=1$ because $\gamma(A)=1$. Then, it should be $F_1(A)=F_2(A)=1$. Therefore, there is a product term γ' of type 1 corresponding to A' such that A<<A' in F_2(Lemma 1(1)), where, by the assumption, $\gamma\neq\gamma'$, that is, $A\neq A'$ holds. Here, $\gamma'(A')=1$ leads to $F_2(A')$ =1 which is equal to $F_1(A')=1$. Therefore, in a similar manner, there is a product term γ'' of type 1 corresponding to A" such that A'<<A" in F_1. Then, γ can be omitted by γ'' because A<<A" and $A\neq A''$, that is, $\gamma\subseteq\gamma''$. Hence, this is contradictory to the assumtion that F_1 is the canonical disjunctive form.

Secondly, assume γ is a product term of type 2 or type 3. Letting A be an element corresponding to γ, $\gamma(A)=1/2$ leads to $F_1(A)=1/2$, because if we assume that $F_1(A)=1$, then the following contradiction arises: there should exist a simple phrase γ' such that $\gamma'(A)=1$ in F_1 and γ is omitted by γ' (Lemma 2(1)). Hence, $F_2(A)=1/2$ holds. This means that there is a product term γ' of type 2 or type 3 corresponding to A' such that A'<<A (Lemma 2(2)) or that there is a simple phrase corresponding to A' such that $A\cap A'\neq\emptyset$ (Lemma 1(2)). Here, the latter does not hold, since if so, then $F_2(A')=F_1(A')=1$ dictates that there is a simple phrase γ'' corresponding to A" such that A'<<A" in F_1 and γ is absorbed by γ''. Therefore, only the former stands valid, where, by the assumption, $\gamma\neq\gamma'$, that is, $A\neq A'$. Similarly, from $\gamma'(A')=1/2$, we can show that there is a product term γ'' of type 2 or type 3 corresponding to A" such that A"<<A' in F_1. Then, γ is absorbed by γ'' because A"<<A and $A''\neq A$. This is contradictory to the assumption that F_1 is a canonical disjunctive form. Therefore, any product term which exists in F_1 also exists in F_2. From the above, we have shown that the canonical disjunctive form of F is determined uniquely. (Q.E.D.)

The following is an algorithm to obtain the canonical disjunctive form of any given regular ternary logic function:
(1) expand the given logic formula into a disjunctive form (a disjunction of product terms),
(2) expand product terms of type 2' and type 3' into the disjunctions of product terms of type 2 and type 3, respectively,
(3) based on the absorption law, omit, if any, product terms which are included by other product terms,
(4) the logic formula obtained finally is a canonical disjunctive form.

[Example 6] The canonical disjunctive form of the regular ternary logic function of Example 4 is obtained as follows:
$$F=\overline{x}_1+(1/2)\cdot\overline{x}_1+(1/2)\cdot x_2$$
$$=\overline{x}_1+(1/2)\cdot\overline{x}_1\cdot x_2+(1/2)\cdot\overline{x}_1\cdot\overline{x}_2+(1/2)\cdot x_1\cdot x_2$$
$$+(1/2)\cdot\overline{x}_1\cdot x_2$$
$$=\overline{x}_1+(1/2)\cdot x_1\cdot x_2.$$

This paper has concentrated on the canonical disjunctive form, but the canonical conjunctive form can also be treated in a similar fashion.

6. Conclusion

We have defined regular ternary logic functions as a significant and useful family of ternary functions and have discussed the fundamental properties of these functions. In particular, we have considered their representations and canonical forms. Recently, Yamamoto[1980] has introduced three-valued majority functions as a family of significant ternary logic functions. The three-valued majority functions are a special example of regular ternary logic functions described in this paper.

References

Kleene, S.C.[1952], Introduction to Metamathematics, North-Holland Pub.,pp.332--340.
Goto,M.[1949],Application of logical mathematics to the theory of relay networks, Jour. IEE Japan, Vol.69.
Mukaidono,M.[1969],On the mathematical structure of C-type fail-safe logic, Trans. IECE Japan, 52-C, 12.
Mukaidono,M.[1972],On the B-ternary logical function -- A ternary logic considering ambiguity, Systems· Computers·Controls, 3, No.3, 27--36.
Mukaidono,M.[1978],The B-ternary logic and its applications to the detection of hazards in combinational switching circuits, Proceedings of 8-th ISMVL, 269--275.
Mukaidono,M.[1980],Some kinds of functional completeness of ternary logic functions, Proceedings of 10-th ISMVL, 81--87.
Mukaidono,M.[1981], A set of independent and complete axioms for a fuzzy algebra (Kleene algebra), Proceedings of 11-th ISMVL, 27--34.
Mukaidono,M.[1982], New canonical forms and their applications to enumerating fuzzy switching functions, Proceedings of 12-th ISMVL, 275--279.
Mukaidono,M.[1978b], A special kinds of ternary logic functions and their applications to fail safe logic circuits, preprint.
Yamamoto,Y. and S. Fujita[1980], Three-valued majority functions, Trans. IECE Japan, Vol.J63-D,No.6.
Yoeli,M. and S.Rinon[1964], Application of ternary algebra to the static hazards, J. ACM, 11, 1.

Session 7A
Reliable Design

PREVIOUS PAGE
IS BLANK

CYCLIC ST-AN CODES AND MODULAR ST DISTANCE

by

Yoshiteru OHKURA,   Ryosaku SHIMADA   and   Toshiharu HASEGAWA

Tokushima Bunri University, Yamashiro, Tokushima, 770 Japan
Faculty of Engineering, Tokushima University, Tokushima, 770 Japan
Faculty of Engineering, Kyoto University, Kyoto, 606 Japan

ABSTRACT

Cyclic AN codes play important roles for detecting and correcting errors in digital systems consisting of arithmetic processors and data transmission channels. This paper describes a new class of ternary cyclic AN codes for the digital systems using the symmetric-ternary numbers. In this class, every code with code length n is an *Ideal* in the finite ring of the absolute-minimum residue class modulo 3^n-1. A new concept of modular distance is introduced for these codes. Many formulas to calculate the actual minimum distance are presented. A good number of multiple-error correcting codes are effectively constructed.

1. INTRODUCTION

Binary arithmetic AN codes are useful for error-detecting and error-correcting in digital systems consisting of arithmetic processors and data trnsmission channels [1]. An arithmetic AN code with cyclic nature is called a cyclic AN code. A class of binary cyclic AN codes with multiple-error correcting capability were first proposed by D. Mandelbaum [2]. Formulas to calculate the minimum distance of the larger class of these codes were given by N. T. Tsao-Wu [3] and R. T. Chien et al. [4]. The theory of cyclic AN codes has been extended to non-binary ones [5,6]. This paper proposes a new class of ternary cyclic AN codes.

For a ternary number N,

$$N = \sum_{i=0}^{n-1} \alpha_i 3^i,$$

there are two important number representations; modulo-3 (M3) representation using 0, 1 and 2 as each coefficient α_i and symmetric-ternary (ST) representation using -1, 0 and +1 as each α_i. Ternary arithmetic operations using the ST representation can be performed without considering distinction between signs of numbers. This makes the ternary arithmetic systems clear and efficient considerably [7]. ST arithmetic AN codes, which were proposed by the authors, are considered to be useful for such systems [8,9].

In this paper, cyclic ST-AN codes are defined as a subclass of ST-AN codes after a brief summary of ST-AN codes. Every cyclic ST-AN code is an *Ideal* in the finite ring of the absolute-minimum residue class modulo 3^n-1. A new type of modular distance* between two integers is defined in that ring. The error-correcting capability of various cyclic ST-AN codes are evaluated in the distance.

2. SUMMARY of ST-AN CODES

An arithmetic AN code is a set of products AN's, where N is an integer to be coded and A (code generator) is a constant positive integer. When every AN (code number or code word) is expressed in ST representation (1),

$$AN = a_{n-1}3^{n-1} + \cdots + a_13 + a_0$$
$$= (a_{n-1} \cdots a_1 a_0)_{ST}, \quad (a_i \in \{1,0,\bar{1}\},$$
$$i=0,1, \cdots ,n-1), \quad (1)$$

where $\bar{1}$ stands for -1, the code is called an ST-AN code. If the code generator A and radix 3 are not relatively prime, the lowest order digits in every code number will always be zeros. This is undesirable, because these digits do not contribute to error control. Therefore, A and 3 should be relatively prime, that is,

$$(A, 3) = 1. \quad (2)$$

ST arithmetic weight, shortly ST weight, of an integer $N=(a_{m-1} \cdots a_1 a_0)_{ST}$ is defined by

$$W_{ST}(N) = \sum_{i=0}^{m-1} |a_i|. \quad (3)$$

The ST weight of every code number AN (or every integer) is easiliy obtained from the number of nonzero digits in its ST representation (1), because the absolute value $|a_i|$ is equal to either 0 or 1, but not 2.

Suppose that, in consequence of failure or noise in the system, an error occurs at only one digit a_i of a code word AN and then AN turns into an erroneous word R. Then, two types of errors are considered as follows:

---

* The ordinary concept of modular distance was first proposed by T. R. N. Rao et al. [11]. This is defined as a distance between two integers in the non-negative minimum residue class modulo 3^n-1.

(1) 1-error ; the difference between AN and R
  $=3^i$, (as shown in Fig. 1(a)).
(2) 2-error ; the difference between AN and R
  $=2\times3^i$, (as shown in Fig. 1(b)).
In the ordinary AN codes, weight 1 is given to both 1-error and 2-error. This means that every error at one digit is treated as the same in the weight. In certain types of arithmetic processors and data transmission channels [9], the probability that 2-error occurs is remarkably less than the probability that 1-error does. In this case, ST weight is a more natural metric than the ordinary one.
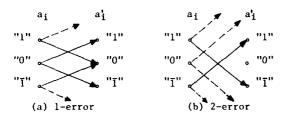
Fig. 1   Types of the errors occuring at a digit a_i in $AN=(a_{n-1} \ldots a_i \ldots a_1 a_0)_{ST}$.
---→ : A carry into the upper digit a_{i+1}, which is a kind of apparent propagation of these errors.)

3. CYCLIC ST-AN CODES and THEIR CONSTRUCTION

[Definition 1] An ST-AN code is a cyclic ST-AN code, if and only if, for any code number;

$$AN = (a_{n-1} \; a_{n-2} \cdots a_1 \; a_0)_{ST},$$

an ST number;

$$(a_{n-2} \cdots a_1 \; a_0 \; a_{n-1})_{ST}$$

obtained by shifting cyclically the digits of AN to the left once is also a code number.

For any code number AN of an ST-AN code, a number obtained by shifting cyclically the digits of AN to the left once is expressed in (4).

$$(a_{n-2} \cdots a_1 \; a_0 \; a_{n-1})_{ST} = 3AN - a_{n-1}(3^n-1) \quad (4)$$

If 3^n-1 is divisible by A, this number is also a multiple of A, that is, another code word. If 3^n-1 is not divisible by A, this number can not be a code word. Therefore, the following theorem is obtained.

[Theorem 1] An ST-AN code with code length n is a cyclic ST-AN code if and only if the code generator A divides 3^n-1.

The above theorem means that there exists a positive integer B satisfying the following equation:

$$3^n - 1 = AB \quad (5)$$

Therefore, a cyclic ST-AN code can be constructed

as follows:
(i) A positive integer B such that $(3,B)=1$ is chosen for the number of code words.
(ii) If 3 belongs to the exponent e modulo B, that is, $e=E(3,B)$, there exists a positive integer A satisfying

$$A = (3^e - 1)/B \quad (6)$$

Then, an ST-AN code with code length n=e is generated by A in (6). This code is a cyclic ST-AN code by Theorem 1.

An absolute-minimum complete-residue system modulo AB $(=3^n-1)$;

$$R_{AB} = \{0,\pm1,\pm2, \ldots ,\pm(\frac{AB}{2}-1),\frac{AB}{2}\} \quad (7)$$

forms a commutative ring under addition and multiplication modulo AB. The cyclic ST-AN code is a subset of R_{AB}, which consists of multiples of A. For any two code words AN_1 and AN_2,

$$(AN_1 \pm AN_2) \bmod AB = A[(N_1 \pm N_2) \bmod B]^*.$$

This means that the sum of the code words (or the difference between them) is also another code word. In this sence, cyclic ST-AN code is a linear code. It can be shown that cyclic ST-AN code is an *Ideal* I_A of R_{AB} generated by A.

The structure of a cyclic ST-AN code I_A with code length n=e is explained through the decomposition of the absolute-minimum complete-residue system modulo B, i. e., R_B. The following discussion is analogous to that of ordinary cyclic AN codes [3,5]. A cyclic ST-AN code can be decomposed as follows:

$$I_A = \sum_{d_j|B} Ad_j \cdot G(B/d_j), \quad (G(B/B)=G(1)=\{0\}, \quad (8)$$

where d_j $(1\le d_j\le B)$ is an exact divisor of B and $G(B/d_j)$ is an absolute-minimum reduced-residue system modulo B/d_j. Then, $G(B/d_j)$ forms a commutative group under multiplication. Every term $Ad_j \cdot G(B/d_j)$ is called a subcode. Each subcode has as many code words as $\varphi(B/d_j)$, i. e., the order of $G(B/d_j)$, where φ is Euler's function.

In every $G(B/d_j)$, the subset;

$$H^1(B/d_j) = \{3^k \bmod B/d_j \mid k=0,1,\ldots,e_j-1\}, \quad (e_j=E(3,B/d_j)) \quad (9)$$

forms a subgroup of $G(B/d_j)$ under multiplication.
Consider subsets formed as follows: The first subset is the subgroup $H^1(B/d_j)$ with the identity. The second subset $H^2(B/d_j)$ is a set which the first element is any element b_2 of $G(B/d_j)$ not appearing in the first subset and the rest elements are obtained by multiplying each subgroup element by the first element b_2. So that each element is given by $[b_2\times(3^k \bmod B/d_j)] \bmod B/d_j = b_2 3^k \bmod B/d_j$. Similarly a therd, fourth and fifth subset are formed and each is formed with a previously unused group element until all the group elements appear somewhere. As the results, $G(B/d_j)$ can be partitioned into mutually exclusive cosets;
* For any two integers a and m>0, the absolute-minimum residue modulo m is denoted "a mod m" throughout this paper.

295

$$H^l(B/d_j) = \{b_l 3^k \bmod B/d_j \mid k=0,1,\cdots,e_j-1\}. \tag{10}$$

A subcode corresponding to $G(B/d_j)$ is moreover partitioned into one or more component codes in (11).

$$A d_j \cdot G(B/d_j) = \sum_{l=1}^{\nu_j} A d_j \cdot H^l(B/d_j),$$
$$(\nu_j = \mathscr{Y}(B/d_j)/e_j) \tag{11}$$

Every component code consists of code words which are cyclic shifts of a code word belonging to it. The component code is called to be strictly cyclic. Evidently, every code word of a component code has exactly the same ST weight. Code word $A d_j b_l$ of a component code $A d_j \cdot H^l(B/d_j)$ is called the leading code word and the ST weight is called ST weight of the component code.

When a code word AN of I_A is expressed in ST representation (1), a digit a_i is given by the following equation [10].

$$a_i = -(B \bmod 3)[(N \times 3^{n-i} \bmod B) \bmod 3],$$
$$(i=0,1,\cdots,n-1) \tag{12}$$

Suppose that code word AN is the leading code word of $A d_j \cdot H^l(B/d_j)$. Then, $N=d_j b_l$ and d_j is an exact divisor of B. Equation (12) is consequently expressed in the following equation;

$$\begin{aligned} a_i &= -(B \bmod 3)[d_j(b_l 3^{n-i} \bmod B/d_j) \bmod 3] \\ &= -(B \bmod 3)(d_j \bmod 3)[(b_l 3^{n-i} \bmod B/d_j) \\ &\qquad \bmod 3]. \end{aligned} \tag{13}$$

The term $b_l 3^{n-i} \bmod B/d_j$ in (13) is an element of the coset $H^l(B/d_j)$. When the integer i of this term changes from 0 to n-1, the value of this term takes all elements of this coset exactly at n/e_j times. Since $(3,B)=1$, B mod 3 $\neq$ 0. So that d_j mod 3 $\neq$ 0 because d_j is an exact divisor of B. Therefore, the digit a_i in (13) is nonzero, i. e., 1 or $\bar{1}$, unless $[(b_l 3^{n-i} \bmod B/d_j) \bmod 3]$ is equal to zero. From the definition of ST weight, the ST weight of the component code is given by the following theorem:

[Theorem 2] ST weight of a component code $A d_j \cdot H^l(B/d_j)$, that is, $W_{ST}(A d_j b_l)$ is given by the following formula:

$$\begin{aligned} W_{ST}(A d_j b_l) = n/e_j \times &\text{[the number of those} \\ &\text{element of } H^l(B/d_j) \text{ which are not} \\ &\text{multiples of 3],} \end{aligned} \tag{14}$$

where $n(=e_1)$ is the code length.

Example 1. A cyclic ST-AN code I_A with 44 code words $(B=44=2^2\times11)$;
   code length n = E(3,44) = 10
   code generator A = 1342 (from (6))
Since B has 6 divisors 1,2,4,11,22 and 44, I_A is decomposed into 6 subcodes as follows;

$$\begin{aligned} I_A &= A \cdot G(44) + A2 \cdot G(22) + A4 \cdot G(11) + A11 \cdot G(4) \\ &\qquad + A22 \cdot G(2) + \{0\} \\ &= A \cdot \{\pm 1, \pm 3, \pm 5, \pm 7, \pm 9, \pm 13, \pm 15, \pm 17, \pm 19, \pm 21\} \end{aligned}$$

$$\begin{aligned} &+ A2 \cdot \{\pm 1, \pm 3, \pm 5, \pm 7, \pm 9\} \\ &+ A4 \cdot \{\pm 1, \pm 2, \pm 3, \pm 4, \pm 5\} \\ &+ A11 \cdot \{\pm 1\} + A22\{1\} + \{0\}. \end{aligned}$$

Then, for instance, a subcode $A2 \cdot G(22)$ is decomposed into two component codes $A2 \cdot H^1(22)$ and $A2 \cdot H^2(22)$, because $\nu_2 = \mathscr{Y}(22)/E(3,22)=10/5$. The subgroup $H^1(22)$ and the other coset $H^2(22)$ are shown in Table 1. The ST weight of the component code $A2 \cdot H^1(22)$ is given by (14). Consequently,

$$W_{ST}(A2) = 10/5 \times 3 = 6 \ (=W_{ST}(-A2)).$$

In fact, both leading code words of these component codes are expressed in

$$A2 = 2684 = (0011\bar{1}\ 0011\bar{1})_{ST}$$
and
$$-A2 = -2684 = (00\bar{1}\bar{1}1\ 00\bar{1}\bar{1}1)_{ST}.$$

Table 1   The subgroup and another coset in $G(22)$

k	0	1	2	3	4	
$3^k \bmod 22$	1	3	9	5	-7	$H^1(22)$
$-3^k \bmod 22$	-1	-3	-9	-5	7	$H^2(22)=-H^1(22)$

4. MODULAR ST DISTANCE and ERROR-CORRECTING CAPABILITY of CYCLIC ST-AN CODES

[Definition 2]   Modular ST distance $D_{MST}(N,M)$ between any pair of integers in R_{AB} is defined as

$$D_{MST}(N,M) = W_{ST}((N-M) \bmod AB). \tag{15}$$

By using the properties of ST weight [8,9] and the above definition, it can be shown that the following theorem is true [10]. Modular ST distance is simply called MST distance, hereafter.

[Theorem 3]   MST distance is a metric function satisfying the following relations.

$$\left. \begin{aligned} D_{MST}(N,M) &\geq 0 \qquad \text{(positive definite)} \\ D_{MST}(N,M) &= D_{MST}(M,N) \ \text{(symmetry)} \\ D_{MST}(N,M) &\leq D_{MST}(N,L) + D_{MST}(L,M) \\ &\qquad\qquad \text{(triangle inequality)} \end{aligned} \right\} \tag{16}$$

[Definition 3]   The minimum of all the distances between every pair of code words of a cyclic ST-AN code is called the minimum MST distance d_m of the code.

As a cyclic ST-AN code is a linear code, the addition or the subtraction modulo AB of every pair of code words results in another code word. Namely, the MST distance between every two code words is equal to ST weight of another code word. Therefore, the minimum MST distance of a cyclic ST-AN code is equal to the minimum ST weight of the code.

Suppose that an ST number E is erroneously added to the code word AN in consequence of failure or noise in the system. Then, if ST

weight of E is d this error is called a d-fold error. The relation between the minimum MST distance of a cyclic ST-AN code and error-correcting capability is described in Theorem 4 and Corollary 4.1. These proofs are omitted because these can be derived by using the triangle inequality in (16), as easily as the case in parity check codes.

[Theorem 4] A cyclic ST-AN code can detect all errors of d-fold or less if and only if the minimum MST distance d_m satisfies $d_m \geq d+1$. And a cyclic ST-AN code can correct all errors of t-fold or less if and only if $d_m \geq 2t+1$.

[Corollary 4.1] A cyclic ST-AN code can correct all errors of t-fold or less and can detect all errors of d-fold or less if and only if $d_m \geq t+d+1$, where $t \leq d$.

5. CALCULATION of MINIMUM MST DISTANCE

A basic way of generating a cyclic ST-AN code has been given in 3. The error-correcting capability is measured by using of the minimum MST distance as mentioned in 4. If the capability of the code is unknown the code is useless. The minimum MST distance can be effectively calculated by the following three steps procedure.
    1. Decomposition of B, the number of code words, as shown in Example 1.
    2. Calculation of ST weight of every compo-nent code by using Theorem 2.
    3. Determination of the minimum among these weights.
Many formulas to calculate the minimum MST distances of cyclic ST-AN codes having the following forms of B will be given. In these forms, both p and q are prime numbers not less than 5 and have various constraints.
    i. p, 2p, 4p
    ii. p^α, $2p^\alpha$, $4p^\alpha$, $(\alpha \geq 2)$
    iii. 2^γ, $(\gamma \geq 3)$
    iv. pq, 2pq
    v. $p^\alpha q$, $2p^\alpha q$, $(\alpha \geq 2)$
    vi. $p^\alpha q^\beta$, $2p^\alpha q^\beta$, $(\alpha, \beta \geq 2)$
Every subcode of the cyclic ST-AN codes mentioned here consists of one component code or two in terms of the constraints. If a subcode consists of a single component code is called to be self-complementary. Then, $\varphi(B/d_j)/e_j=1$. From (11), $G(B/d_j) = H^i(B/d_j)$ because 3 is the primitive root modulo B/d_j. Therefore, if a code word AN belongs to the component code, -AN also belongs to the same component code. On the other hand, if a subcode consists of two component codes and each AN and -AN belongs to another component code, as shown in $A2 \cdot G(22)$ of Example 1. the component codes are called to be mutually complementary. In these cases, every code word belonging to the subcode has the same weight. Therefore,
 ST weight of a self-complementary component code $Ad_j \cdot G(B/d_j)$:
 $n/e_j \times$ [the number of elements which are not multiples of 3 in $G(B/d_j)$]
and
 ST weight of mutually complementary component codes $\pm Ad_j \cdot H^i(B/d_j)$:

$n/e_j \times$ [a half of the number of elements which are not multiples of 3 in $G(B/d_j)$].
The following formulas to calculate the minimum MST distance can be obtained on the basis of these points. They are derived from number theory. Since these processes are pretty long, an example of the fundamental derivation will be given in Appendix. The other formulas can be obtained by applying the simmilar way to the case of Appendix.

i. B=p, 2p, 4p
i-1. p; a prime having 3 as a primitive root

(a) B=p; n=p-1, $A=(3^n-1)/p$
$$d_m = \begin{cases} 2/3 \times (p-1), & (p \equiv 1 \pmod 3) \\ 2/3 \times (p+1), & (p \equiv -1 \pmod 3) \end{cases}$$

(b) B=2p; n=p-1, $A=(3^n-1)/(2p)$
$$d_m = \begin{cases} 2/3 \times (p-1), & (p \equiv 1 \pmod 3) \\ 2/3 \times (p-2), & (p \equiv -1 \pmod 3) \end{cases}$$

(c) B=4p, $4|(p-1)$; n=p-1, $A=(3^n-1)/(4p)$
 $d_m = 2/3 \times (p-2)$, $(p \equiv -1 \pmod 3)$

i-2. p; a prime having -3, but not 3, as a primitive root

(a') B=p; n=(p-1)/2, $A=(3^n-1)/p$
 $d_m = 1/3 \times (p+1)$, $(p \equiv -1 \pmod 3)$

(b') B=2p; n=(p-1)/2, $A=(3^n-1)/(2p)$
 $d_m = 1/3 \times (p-2)$, $(p \equiv -1 \pmod 3)$

(c') B=4p; n=p-1, $A=(3^n-1)/(4p)$
 (See Example 1.)
 d_m = (the same as d_m in (c))

ii. B=p^α, $2p^\alpha$, $4p^\alpha$, $(\alpha \geq 2)$
ii-1. p^α; a power of a prime having 3 as a primitive root

(d) B=p^α; n=$p^{\alpha-1}(p-1)$, $A=(3^n-1)/p^\alpha$
$$d_m = \begin{cases} 2/3 \times p^{\alpha-1}(p-1), & (p \equiv 1 \pmod 3) \\ 2/3 \times p^{\alpha-2}(p+1)(p-2), & (p \equiv -1 \pmod 3) \end{cases}$$

(e) B=$2p^\alpha$; n=$p^{\alpha-1}(p-1)$, $A=(3^n-1)/(2p^\alpha)$
$$d_m = \begin{cases} 2/3 \times p^{\alpha-1}(p-1), & (p \equiv 1 \pmod 3) \\ 2/3 \times p^{\alpha-1}(p-2), & (p \equiv -1 \pmod 3) \end{cases}$$

(f) B=$4p^\alpha$, $4|(p-1)$; n=$p^{\alpha-1}(p-1)$, $A=(3^n-1)/(4p^\alpha)$
 $d_m = 2/3 \times p^{\alpha-1}(p-2)$, $(p \equiv -1 \pmod 3)$

ii-2. p^α; a power of a prime having -3, but not 3, as a primitive root

(d') B=p^α; n=$p^{\alpha-1}(p-1)/2$, $A=(3^n-1)/p^\alpha$
 $d_m = 1/3 \times p^{\alpha-2}(p+1)(p-2)$, $(p \equiv -1 \pmod 3)$

(e') B=$2p^\alpha$; n=$p^{\alpha-1}(p-1)/2$, $A=(3^n-1)/(2p^\alpha)$
 $d_m = 1/3 \times p^{\alpha-1}(p-2)$, $(p \equiv -1 \pmod 3)$

(f') B=$4p^\alpha$; n=$p^{\alpha-1}(p-1)$, $A=(3^n-1)/(4p^\alpha)$
 d_m = (the same as d_m in (f))

iii. B=2^γ, $(\gamma \geq 3)$

(g) B=2^γ; n=$2^{\gamma-2}$, $A=(3^n-1)/(2^\gamma)$
 $d_m = 2^{\gamma-3}$ $(= 1/8 \times 2^\gamma)$

297

iv. **B=pq, 2pq**
iv-1. p, q; primes having 3 as each primitive
root,
$4|(p-1)$ or $4|(q-1)$,
$(p-1, q-1)=2$.

(h) $B=pq$; $n=(p-1)(q-1)/2$, $A=(3^n-1)/(pq)$

$$d_m = \begin{cases} 1/3\times[(p-1)(q-1)-4], & (p\equiv q\equiv-1 \ (\mathrm{mod}\ 3)) \\ 1/3\times(p-1)(q-1), & (\text{the other cases}) \end{cases}$$

(i) $B=2pq$; $n=(p-1)(q-1)/2$, $A=(3^n-1)/(2pq)$

$$d_m = \begin{cases} 1/3\times(p-1)(q-2), & (p\equiv-q\equiv1 \text{ or} \\ & p\equiv q\equiv-1 \ (\mathrm{mod}\ 3),\ p>q) \\ 1/3\times(p-2)(q-1), & (p\equiv-q\equiv-1 \text{ or} \\ & p\equiv q\equiv-1 \ (\mathrm{mod}\ 3),\ p<q) \end{cases}$$

iv-2. p; a prime having 3 as a primitive root,
q; a prime having -3, but not 3,
$(p-1, q-1)=2$.

(h') $B=pq$; (all the same as in (h))

(i') $B=2pq$; $n=(p-1)(q-1)/2$, $A=(3^n-1)/(2pq)$

$$d_m = \begin{cases} 1/3\times(p-1)(q-2), & (p\equiv-q\equiv1 \text{ or} \\ & p\equiv q\equiv-1 \ (mod\ 3),\ p>q) \\ 1/3\times(p-2)(q-1), & (p\equiv q\equiv-1 \ (mod\ 3),\ p<q) \end{cases}$$

v. **B=$p^\alpha q$, $2p^\alpha q$, ($\alpha\geq2$)**
v-1. p^α, q; a power of a prime and a prime
having 3 as primitive roots,
$4|(p-1)$ or $4|(q-1)$,
$(p^\alpha-p^{\alpha-1}, q-1)=2$.

(j) $B=p^\alpha q$; $n=p^{\alpha-1}(p-1)(q-1)/2$, $A=(3^n-1)/(p^\alpha q)$

$$d_m = \begin{cases} 1/3\times p^{\alpha-1}(p-1)(q-1), & (p\equiv1 \ (\mathrm{mod}\ 3)) \\ 1/3\times p^{\alpha-2}(p+1)(p-2)(q-1), & (p\equiv-q\equiv-1 \text{ or} \\ & p\equiv q\equiv-1,\ 2p+1<q) \\ 1/3\times p^{\alpha-1}[(p-1)(q-1)-4], & (p\equiv q\equiv-1,\ 2p+1>q) \end{cases}$$

(k) $B=2p^\alpha q$; $n=p^{\alpha-1}(p-1)(q-1)/2$,

$A=(3^n-1)/(2p^\alpha q)$

$$d_m = \begin{cases} 1/3\times p^{\alpha-1}(p-1)(q-2), & (p\equiv-q\equiv1 \text{ or} \\ & p\equiv q\equiv-1,\ p>q) \\ 1/3\times p^{\alpha-1}(p-2)(q-1), & (p\equiv q\equiv-1,\ p<q \text{ or} \\ & p\equiv-q\equiv-1) \end{cases}$$

v-2. p^α; a power of a prime having 3 as a
primitive root,
q ; a prime having -3, but not 3,
$(p^\alpha-p^{\alpha-1}, q-1)=2$.

(j') $B=p^\alpha q$; (all the same as in (j))

(k') $B=2p^\alpha q$; (all the same as in (k))

v-3. p^α; a power of a prime having -3, but not
3 as a primitive root,
q ; a prime having 3,
$(p^\alpha-p^{\alpha-1}, q-1)=2$.

(j") $B=p^\alpha q$; (all the same as in (j))

(k") $B=2p^\alpha q$; (all the same as in (k))

vi. **B=$p^\alpha q^\beta$, $2p^\alpha q^\beta$, ($\alpha,\beta\geq2$)**
vi-1. p^α, q^β; powers of primes having 3 as
primitive roots,
$4|(p-1)$ or $4|(q-1)$,
$(p^\alpha-p^{\alpha-1}, q^\beta-q^{\beta-1})=2$

(l) $B=p^\alpha q^\beta$; $n=p^{\alpha-1}q^{\beta-1}(p-1)(q-1)/2$,

$A=(3^n-1)/(p^\alpha q^\beta)$

$$d_m = \begin{cases} 1/3\times p^{\alpha-1}q^{\beta-2}(p-1)(q+1)(q-2), & (p\equiv-q\equiv1 \text{ or} \\ & p\equiv q\equiv-1,\ q<(p-1)/2) \\ 1/3\times p^{\alpha-2}q^{\beta-1}(p+1)(p-2)(q-1), & (p\equiv-q\equiv-1 \text{ or} \\ & p\equiv q\equiv-1,\ q>2p+1) \\ 1/3\times p^{\alpha-1}q^{\beta-1}[(p-1)(q-1)-4], & (p\equiv q\equiv-1, \\ & (p-1)/2<q<2p+1) \end{cases}$$

(m) $B=2p^\alpha q^\beta$; $n=p^{\alpha-1}q^{\beta-1}(p-1)(q-1)/2$,

$A=(3^n-1)/(2p^\alpha q^\beta)$

$$d_m = \begin{cases} 1/3\times p^{\alpha-1}q^{\beta-1}(p-1)(q-2), & (p\equiv-q\equiv1 \text{ or} \\ & p\equiv q\equiv-1,\ p<q) \\ 1/3\times p^{\alpha-1}q^{\beta-1}(p-2)(q-1), & (p\equiv q\equiv-1,\ p>q \text{ or} \\ & p\equiv-q\equiv-1) \end{cases}$$

vi-2. p^α; a power of a prime having 3 as a
primitive root,
q^β; a power of a prime having -3, but not
3,
$(p^\alpha p^{\alpha-1}, q^\beta q^{\beta-1})=2$.

(l') $B=p^\alpha q^\beta$; (all the same as in (l))

(m') $B=2p^\alpha q^\beta$; (all the same as in (m))

Table 2 Minimum MST distance of cyclic ST-AN codes
(°; minimum B satisfying each type of conditions)

B	ψ(B)	E(3,B)=n	E(-3,B)	Type	d_m
5	4	4	4	(a)°	4
7	6	6	3	(a)	4
8	4	2	2	(g)°	1
10	4	2	2	(b)°	2
11	10	5	10	(a')°	4
13	12	3	6	(a)	2
14	6	6	3	(b)	4
16	8	4	4	(g)	2
17	16	16	16	(a)	12
19	18	18	9	(a)	12
20	8	4	4	(c)°	2
22	10	5	10	(b')°	3
23	22	11	22	(a')	8
25	20	20	20	(d)°	12
26	12	3	6	(b)	1
28	12	6	3	(b)	2
29	28	28	28	(a)	20
31	30	30	15	(a)	20
32	16	8	8	(c)	4
34	16	16	16	(b)	12
35	24	12	12	(h)°	8
37	36	18	9	(h)	12
40	16	4	4		2
41	40	8	8		4
43	42	42	21	(a)	28
44	20	10	10	(c')°	6
46	22	11	22	(b')	7
47	46	23	46	(a')	16
49	42	42	21	(d)	28
50	20	20	20	(e)°	10
52	24	6	6		2
53	52	52	52	(a)	36
55	40	20	20	(h')°	12
64	32	16	16	(g)	8
70	24	12	12	(i)°	6
100	40	20	20	(f)°	10
110	40	20	20	(i')°	10
175	120	60	60	(j)°	36
350	120	60	60	(k)°	30
529	506	253	506	(d')°	168
539	420	210	210	(j')°	140
1058	506	253	506	(e')°	161
1078	420	210	210	(k')°	126
1225	840	420	420	(l)°	252
2116	1012	506	506	(f')°	322
2450	840	420	420	(l')°	210
2645	2024	1012	1012	(j")°	644
5290	2024	1012	1012	(k")°	506

Minimum MST distance d_m of a code with composite number B not satisfying the conditions mentioned above can be found by a computer program performing the basic process to find d_m (Step 1 –

3). A part of the results is shown in Table 2. The codes whose d_m's can be found by the formulas mentioned above are also shown in that table.

The minimum MST distance d_m of each code mentioned above is plotted against B in Fig. 2. It can be seen that d_m of every code lies along one of four lines, i. e., $d_m/B=2/3$, $1/3$, $1/6$ or $1/8$. From every formula to calculate d_m, it can be found that d_m/n, which is called the capability of the code, is nearly equal to 2/3 except a case of (g). The capability of a code satisfying the condition (g) whose B is represented as 2^γ is incidentally equal to 1/2.

These cyclic ST-AN codes are multiple-error correcting codes and are effectively useful because of simplicity of these formulas.
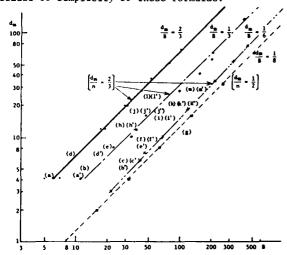
Fig. 2 Plot of d_m for cyclic ST-AN codes satisfying the conditions i - vi on B

6. CONCLUSION

A new class of ternary cyclic AN codes and a new type of modular arithmetic distance were introduced in this paper. This class is considered to be useful for error control in ternary digital systems consisting of symmetric-ternary arithmetic processors and ternary data transmission channels.

Symmetric-ternary arithmetic operations can be performed without considering distinction between signs of numbers. These operations are much simpler and clearer than the ordinary ternary ones in which a negative number is represented as the complement to 3^n or 3^n-1. This makes the derivation of these formulas for d_m effective in many cases mentioned in 5.

Many formulas to calculate the actual minimum MST distances of cyclic ST-AN codes were presented under various conditions on prime factors of B. Cyclic ST-AN codes satisfying these conditions have multiple-error correcting capability. The decoding problem will be studied in subsequent works.

REFERENCES

[1] Peterson, W. W. and Weldon Jr., E. J., "Error Correcting Codes", 2nd ed., The MIT Press, Chambridge, Mass. (1972)

[2] Mandelbaum, D., "Arithmetic codes with large distance", IEEE Trans., Inf. Theory, vol. IT-13, pp. 237-242 (1967)

[3] Tsao-Wu, N. T., "Arithmetic cyclic codes", Part I of Communication Theory Group Report No. 10, Northeastern Univ., Boston, Mass. (1968)

[4] Chien, R. T., Hong, S. J. and Preparata, F. P., "Some results in the theory of arithmetic codes", R-440, Coordinated Science Laboratory, Univ. of Illinois (1971)

[5] Fukumura, T. and Goto, M., "Arithmetic Coding Theory", Corona Publishing Co. Ltd, Tokyo (1978) (in Japanese)

[6] Ecker, A., "How to compute the minimum distance for cyclic AN-codes over an arbitrary base", Inf. and Cont., 46, pp. 219-240 (1980)

[7] Mine, H., Hasegawa, T. and Shimada, R., "Ternary four arithmetic operations", IECE Trans., vol. 54-C, No. 1, pp. 66-73 (1971)

[8] Ohkura, Y., Shimada, R. and Hasegawa, T., "Symmetric ternary arithmetic weight and symmetric ternary arithmetic AN codes", 11th Int. Symp. on Multi-Valued-Logic, pp. 163-167 (1981)

[9] Ohkura, Y., Shimada, R. and Hasegawa, T., "Symmetric ternary arithmetic AN codes", IECE Trans., vol. J64-D, No. 6, pp. 502-509 (1981)

[10] Ohkura, Y., Shimada, R. and Hasegawa, T., "Fundamental concept of cyclic ST-AN codes", IECE Trans., vol. J65-D, No. 11, pp. 1358-1365 (1982)

[11] Rao, T. R. N. and Garcia, O. N., "Cyclic and multiresidue codes for arithmetic operations", IEEE Trans., Inf. Theory, vol. IT-17, No. 1, pp. 85-91 (1971)

APPENDIX

A derivation of formulas to calculate the minimum MST distance in the case of i-1 (a) :

Since 3 is a primitive root modulo p and p is a prime number,

$$E(3,p) = \varphi(p) = p-1.$$

Therefore, the code length n = p-1 and the code generator A = $(3^{p-1}-1)/p$ from (6). Moreover,

$$G(p) = H^1(p)$$
$$= \{3^k \bmod p \mid k=0,1,\cdots,p-2\}$$
$$= \{\pm k' \mid k'=1,2,\cdots,(p-1)/2\}.$$

By Theorem 2, ST weight of the component code A· $H^1(p)$ is equal to the number of those elements of $G(p)=H^1(p)$ which are not multiples of 3. When $p\equiv1$ (mod 3), (p-1)/2 is also a multiple of 3. All of the elements which are not multiples of 3 are as follows;

$$\{\pm1,\pm2,\ \pm4,\pm5,\ \cdots\ ,\pm[(p-1)/2-2],\pm[(p-1)/2-1]\}.$$

Then, the number of these elements is given by $2\times(2/3)\times(p-1)/2=2/3\times(p-1)$. When $p\equiv-1$ (mod 3), (p+1)/2 is a multiple of 3. Since all the elements which are not multiples of 3 are as follows;

$$\{\pm1,\pm2,\ \pm4,\pm5,\ \cdots\ ,\pm[(p+1)/2-2],\pm[(p+1)/2-1]\},$$

the number of these elements is $2\times(2/3)\times(p+1)/2= 2/3\times(p+1)$.

A UNIFIED APPROACH TO COMPOSITE MVL WITH MONOTONIC SUBFUNCTION

Matsuroh Nakamichi and Hideo Itoh

Department of Electronic Engineering, Faculty of Engineering
Chiba University, Yayoi-cho, Chiba 260, Japan

ABSTRACT

The concept of composite multiple-valued logic (CMVL) is introduced. The CMVL has subfunctions in the subsets of logical values. The subfunctions, especially monotonic ones play role to make the system testable ; fault detectable and diagonosable.

The CMVL contains C-type fail safe logic and A-type which have one monotonically increasing sub-function, and also contains some new logics, D-type with two monotonic ones, J-type with three monotonic ones etc..

Classification and properties of these functions and systems are discussed using the concept of CMVL, and the properties give suggestions for design strategies of two valued or binary and multiple valued or higher radix testable logical systems.

I. INTRODUCTION

An advantage or merit of multiple-valued logc (MVL) is that it has three or more radix. The higher radix should bring about higher information processing capability per unit gate and also higher transmission density of information per line. To utilize and realize these features, the basic and application oriented studies of MVL have been done [1-4].

Another merit originated in higher radix or multiple logical values is that MVL can have useful two valued and, in general, multiple valued special subfunctions in the subset of logical values. Taking notice of these subfunctions, one can introduce new classes of special functions which are connected with new applications [5-16].

Typical works of this area are found in the studies of fail safe logical systems. Their features are based on the monotonic properties [4], which also bring about testability ; fault detectability and diagnosability [5-12]. Non-fail safe but testable systems and their elements are also studied [12-15], and they are classified using ordered graph [14]. Another approach, the higher radix technique, for testable system are proposed [16].

In this paper, a unified approach to CMVL with monotonic subfunctions is presented. These CMVL's contain C-type and A-type 3 valued logic and D-type, J-type MVL. The relations among these functions and the special properties which are attributed to the monotonic subfunctions are discussed. These properties also give suggestions for designing of testable digital systems [11-15].

II. DEFINITION AND CLASSIFICATION OF CMVL

(1) CMVL and subfunctions

(a) Definition and notations

The MVL can have some non-constant subfunctions in the domain of subsets of multiple logical values. These MVL's are defined as follows :

Definition 1

Let the function of r valued n variable combinational circuit be

$$f : L^n \longrightarrow L , \quad L = \{v_1, v_2, ---v_j, ---v_r\}$$
$$= \{v_{11}, v_{12}, ---v_{1j}, ---v_{1p},$$
$$v_{21}, v_{22}, ---v_{2j}, ---v_{2q},$$
$$v_{i1}, v_{i2}, ---v_{ij}, ---v_{it},$$
$$v_{m1}, v_{m2}, ---v_{mj}, ---v_{mu}\}$$
$$(1)$$
$$p + q --- + t --- + u = r$$

where L is the whole set of v_j or v_{ij} which represent a logical value.

The function f is defined as a composite multiple valued logic (CMVL) or CMVL function , if f has following subfunctions g_i or both g_i and h_j:

$$g_i : L_i^n \longrightarrow L_i , \quad L_i = \{v_{11}, v_{12}, ---v_{1j}, ---v_{it}\} \quad (2)$$

$$h_j : L_j^n \longrightarrow L_j , \quad L_j = \{v_{1j}, v_{2j}, ---v_{ij}, ---v_{mj}\} \quad (3)$$

where L_i and L_j are subsets of logical values, and

$v_{11}, v_{12}, ---v_{ij}, ---v_{it}$ are logical values in the L_i ,

$v_{1j}, v_{2j}, ---v_{ij}, ---v_{mj}$ are the ones in the subsets

$L_1, L_2, ---L_i, ---L_m$ respectively.

By exchanging the partial or all input or output variables to the other definite ones, if the function satisfies the relation (2), or relations (2) and (3) , the function f can be treated as the equivalent CMVL.

300

If some functions in the function group h_j ; $h_1, h_2, ---h_j---$, are the same or the same class of functions, the group of function h_j can be expressed as follows :

$$h_s:L_s^n \longrightarrow L_s, \quad L_s=\{\{S_1\},\{S_2\},--\{S_i\},--\{S_m\}\}\subseteq L \quad (4)$$

or

$$h_p:L_p^n \longrightarrow L_p, \quad L_p=\{\{L_1\},\{L_2\},--\{L_i\},--\{L_m\}\}= L \quad (5)$$

where L_s and L_p are sets of sets, $\{S_i\}\subseteq\{L_i\}$ for every i , and S_i, L_i must satisfy equation (2).

In other word, equation (4),(5) show that the logical operations for the combinations of each logical value in differant $\{S_i\}$ or $\{L_i\}$ are equal or in the same class. If the logical operations are the same, the $\{S_i\}$ or $\{L_i\}$ in equation (4) or (5) can be treated as aequivalent single value with respect to the logical operations. (see Appendix)

(b) Classification by subfunctions [15]

The CMVL is classified by the number and classes of the subfunctions, g_i and h_j , and their properties. Some examples of composite three valued logic (abreviated to CTVL) are shown as follows:

(i) Number of logical values subset

Example 1 Three valued logic with logical values 0,1,and R can have a set in the following combinations of logical values:

$\{1,0\}$,	$\{\{R\},\{1,0\}\}$
$\{0,R\}$,	$\{\{1\},\{0,R\}\}$
$\{R,1\}$,	$\{\{0\},\{R,1\}\}$

Note:In reference[15] and others, R is expressed by symbol Φ.

(ii)Set of subfunctions and related notations

The CTVL can have a set of subfunctions, 2 valued functional complete [4] and monotonic [4] ones etc..

Example 2 A CTVL has following set of 2 valued and equivalent 2 valued subfunctions [15]:

$$g_0 \text{ [FC]} : \{1,0\}^n \longrightarrow \{1,0\}$$
$$h_0 \text{ [MI]} : \{\{R\},\{1,0\}\}^n \longrightarrow \{\{R\},\{1,0\}\} \quad (6)$$

where g_0 [FC] denotes the functional completeness and h_0 [MI] in equation (6) denotes the monotonically increasing for R, and the R is placed first in the $\{\ \}$. Another example is shown as follows:

$$g_1 \text{ or } h_1 \text{[AND]} : \{1,0\}^n \longrightarrow \{1,0\}$$
$$g_2 \text{ or } h_2 \text{[AND]} : \{R,1\}^n \longrightarrow \{R,1\} \quad (7)$$
$$g_3 \text{ or } h_3 \text{[AND]} : \{0,R\}^n \longrightarrow \{0,R\}$$

where g_i or h_i [AND] express the AND function with respect to the first logical value in $\{\ \}$.

(2) CMVL systems and logical elements

(a) Definition and construction rule

Definition 2

The CMVL system is defined as the system (a set of logical gates) with functional completeness and also (useful) subfunctions, or as combinational circuits or networks which are constructed so as to have the subfunctions.

The CMVL system with monotonic subfunctions are constructed using following rule.

Theorem 1

The CMVL system with monotonic subfunctions are constructed with the set of CMVL logical gates with monotonic subfunctions in the same subset of logical values.

If the subfunctions of the gates (elements) are

monotonically increasing for the same logical value or set of values, the CMVL circuits constructed with the elements, and with any structure and any function with respect to the logical values of functional complete subset, have the same monotonically increasing subfunction.

Proof: It is derived from definition 2 and property of monotonically increasing function [15].

Property 1

Any function of the part of the network which is composed of the gates(elements) with functional complete and monotonically increasing subfunctions, has the same monotonically increasing subfunctions.

This property of network simplify the fault detection[17-19].

(c) Family of elements and ordered grph

Minimum unit of network is logical gate or element , and the elements having functional complete and monotonically increasing subfunctions are also a system with the same subfunctions.

The 3 valued elements which can be these systems, and others are classified using ordered graph in Fig. 1 [14].

Fig. 1 Classification of some ternary logical elements
* ( ) :number of elements in each family

301

Definition 3 [14]

Ordered graph is a graphical representation of logical element function f, and it is specified as follows:

(i) node : The input value x_i is marked in node circle, and output value y_i can be marked outside the node with short line to it. The y_i is specified by the output for all x_i inputs as follows:

$$y_i = f(x_i, x_i, ---x_i) = f(x_i) \qquad (8)$$

(ii) line : The line between nodes marked with x_i and x_j, shows the relation of order or advantage between x_i and x_j for determining the output when x_i and x_j applied together.

(iii) directed line : If the line directs to the node x_j from x_i , x_i is sensitizing (line) value[4] and x_j is non-sensitizing one i.e. advantageous one, then the output becomes as follows:

$$f(x_i, x_j) = f (x_j) = y_j \qquad (9)$$

(iv) no line : If there exist no line between nodes, x_i and x_j, the above output for x_i and x_j changes as follows:

$$f(x_i, x_j) = y_k = f(x_k) \qquad (10)$$

where y_k, x_k are another output and input value different from y_i, y_j and x_i , x_j. (example of this case will be shown in Table 2)

Example 3

(i) 2 valued logics

The AND,NAND,OR and NOR functions are represented by ordered graph as shown in Fig.2.

(a) AND　　　　(c) NAND　(e)AND,NAND

(b) OR　　　　　(d) NOR　(f)OR,NOR　(g)AND,NAND

$v_1 v_2$: two different logical values　　OR ,NOR

Fig.2 Examples of ordered graph

(ii) 3 valued logics

Some 3 valued elements under the condition $f(R)=$ are shown in Fig. 1. The elements with functional complete and monotonically increasing for R are C-type ,A-type,Φ -type and J-type in family I, so they can be a CTVL system. The J-type and all types of family II,III and IV except D-type can not have definite output value for $f(0,1,R)$, so the input number must be restricted to 2.

The logical operations of A, C, D,and J-type are also shown in Table 1-3.

III. TESTABLE CTVL SYSTEMS AND SPECIAL FUNCTION

(1)CTVL with 2 valued functional completenes

The C-type [9] and A-type [12] systems are defined as a special class of CTVL with monotonically increasing subfunctions.

Definition 4

The C-type and A-type systems are defined one of the special CTVL with following set of subfunctions g_0 and h_p:

$$g_0[FC] : \qquad \{1,0\}^n \longrightarrow \{1,0\}$$

$$h_p[OR \text{ or } AND] :\{\{R\},\{1,0\}\}^n \longrightarrow \{\{R\},\{1,0\}\} \qquad (11)$$

Example 4 [13-15]

From theorem 1 and equation (11) the elements which can construct the A-type and C-type systems are introduced.

They are shown in Table 1.

Table 1　C-type and A-type elements (NAND)

C-type			A-type		
Truth Table		OrderedGraph	Truth Table		OrderedGraph

The functions of NOR gate can be understood by dividing to subfunctions and combining them.

Property 2

(i) The systems have following properties which come from the AND or OR subfunctions:

$$C : f (1,0,R)=h_p\{\{R\},\{1,0\}\}=R$$
$$A : f (1,0,R)=h_p\{\{R\},\{1,0\}\}=\{1,0\}=b(1 \text{ or } 0) \qquad (12)$$

where $f(1,0,R)$ denotes a function of variables, 1,0, and R, $h_p\{\{R\},\{1,0\}\}$ an equivalent 2 valued subfunction of variables $\{R\}$ and $\{1,0\}$.

Utilizing these properties, in a tree like network, we can have functional independent universal n+1 test as follows [12,15] :

$$C : bb---b, Rb---b, bRb---b, ---, bb---bR$$
$$A : RR---R, bR---R, RbR---R, ---, RR---Rb$$

It is known that we can have a tree like network for any function g_0 or equivalent tree like ones by providing observing or controlling points in tne reconvergent paths [20].

(ii) The C-type has fail safe property, but the A-type has not . The difference also comes from the difference of each element or system subfunctions which are shown in Table 1 or in the following relations:

$$C : f(R,0)=f(R,R)=R \qquad f(R,1)=f(R,R)=R$$
$$A : f(R,0)=f(1,0)\neq f(1,1) \qquad f(R,1)=f(1,1)\neq f(0,1)$$

$$(13)$$

(2) D-type and J-type CTVL functions

(a) D-type and related ones

A CTVL function which is not functional complete but has two monotonic subfunctions is introduced.

Definition 5 [15]

The D-type function is defined as a CTVL function with following two subfunctions h_2 and h_3:

$$h_2[OR] : \{R,1\}^n \longrightarrow \{R,1\}$$
$$h_3[OR] : \{R,0\}^n \longrightarrow \{R,0\} \qquad (14)$$

If the function h_2, h_3 are all AND functions, the CTVL is also defined as E-type.

302

Example 5 [14,15]

The CTVL circuit with above subfunctions is realized using the elements shown in Table 2.

Table 2 D-type and E-type elements

D-type

Truth Table:

	0	1	R
0	0	R	R
1	R	1	R
R	R	R	R

OrderedGraph: (0) (1) → (R), R

E-type

Truth Table:

	0	1	R
0	0	R	0
1	R	1	1
R	0	1	R

OrderedGraph: (0) (1) → (R), R

Property 3 [15]

The D-type function has following properties:
(i) Function value is 1 (0) if and only if the vector of variables is $\mathbf{1}$ ($\mathbf{0}$) as follows:

$$f(\mathbf{1}) = h_2(\mathbf{1}) = 1 \qquad \mathbf{1} = 1,1,\text{---}1$$
$$f(\mathbf{0}) = h_3(\mathbf{0}) = 0 \qquad \mathbf{0} = 0,0,\text{---}0 \qquad (15)$$
$$f(\mathbf{x} \neq \mathbf{1}, \mathbf{x} \neq \mathbf{0}) = R$$

This property can be called "double degeneration".
(ii) All stuck at 0,1 and R faults, single and multiple, in the circuits constructed with D-type elements are detectable with only 2 universal test $\mathbf{1}$ and $\mathbf{0}$ [15,16].

(b) J-type CTVL function with 3 AND or OR sub-functions

The J-type CTVL function is defined as follows:

Definition 6

The J-type function is defined as a CTVL function with following three subfunctions g_1, g_2, and g_3 specified in equation (16) :

$$g_1 \text{ [AND or OR] } : \{1,0\}^n \longrightarrow \{1,0\}$$
$$g_2 \text{ [AND or OR] } : \{R,1\}^n \longrightarrow \{R,1\} \qquad (16)$$
$$g_3 \text{ [AND or OR] } : \{0,R\}^n \longrightarrow \{0,R\}$$

where g_1, g_2, and g_3 are all AND (OR) functions, and they are subdivided to J_1-type with AND , J_2-type with OR subfunctions.

Example 6 [13-15]

The J-type function is realized in the circuits composed of the 2 input gates specified by Table 3.

Table 3 J-type CTVL elements

J_1-type

Truth Table:

	0	1	R
0	0	0	R
1	0	1	1
R	R	1	R

OrderedGraph: (0) ← (1), (R) → R

J_2-type

Truth Table:

	0	1	R
0	0	1	0
1	1	1	R
R	0	R	R

OrderedGraph: (0) → (1), (R) → R

Property 4

The J-type has following properties :
(i) The J-type has following cyclic property.

$$J_1 \begin{cases} f(1,0)=g_1(1,0)=0 \\ f(R,1)=g_2(R,1)=1 \\ f(0,R)=g_3(0,R)=R \end{cases} \quad J_2 \begin{cases} f(1,0)=g_1(1,0)=1 \\ f(R,1)=g_2(R,1)=R \\ f(0,R)=g_3(0,R)=0 \end{cases} \quad (17)$$

(ii) All stuck at 1,0 and R faults of the network composed of the elements shown in Table 3 are detectable with the test set $\mathbf{1}$, $\mathbf{0}$ and $\mathbf{R}$ [13,15].

IV. SOME TESTABLE CMVL SYSTEMS

(1) Testable CMVL systems with 2 redundant logical values

(a) D-type CMVL systems

The D-type systems with D-type CTVL subfunction are proposed.

Definition 7 [15]

The D-type system is defined as a CMVL system with a D-type CTVL and r-2 valued functional completeness as follows:

$$g_o[FC] : L_o^n \longrightarrow L_o \qquad L=\{v_1, v_2, \text{---} v_{r-2}\}$$
$$h_p[D] : \{\{L_o\}, \{R_1\}, \{R_2\}\}^n \longrightarrow \{\{L_o\}, \{R_1\}, \{R_2\}\} \quad (18)$$

where $g_2[D]$ is the D-type CTVL subfunction which has following two OR subfunctions for L_o specified in equation (19).

$$h_{s1}[OR] : \{\{L_o\}, \{R_1\}\}^n \longrightarrow \{\{L_o\}, \{R_1\}\}$$
$$h_{s2}[OR] : \{\{L_o\}, \{R_1\}\}^n \longrightarrow \{\{L_o\}, \{R_2\}\} \quad (19)$$

These functions are derived by exchanging the variable 1 by R_1, 0 by R_2 and R by L_o in equation (14).

Example 7

The D-type 4 valued system (2 valued functional completeness) can be constructed with the element shown in Table 4.

Table 4 D-type CMVL (4 valued) element

Truth Table					Ordered Graph	Note
	0	1	R_1	R_2	R_1 R_2	b : 1 or 0
0	1	0	b	b	(R₁) (R₂)	subclassified by
1	0	0	b	b		
R_1	b	b	R_1	b	{1,0}	$f(R_1, R_2)$
R_2	b	b	b	R_2	{1,0}	$f(1, R_1)$ $f(0, R_1)$ $f(1, R_2)$ $f(0, R_2)$

Property 5

All stuck at definite logical value $(v_1, v_2 \text{---} R_1, R_2)$ faults are detected by applying the dual universal tests $\mathbf{R_1}$ and $\mathbf{R_2}$. If the output is R_1 for $\mathbf{R_1}$, and R_2 for $\mathbf{R_2}$ the system is fault free, if the output is in L_o, the system is fault.

This result corresponds to a proposal of reference [16].

(b) J-type CMVL systems with CTVL subfunction

303

Definition 8 [13,15]

The J-type system is defined as the CMVL system with following subfunctions:

$$g_0[FC] : L_o^n \longrightarrow L_o, \quad L_o = \{v_1, v_2, ---v_{r-2}\} \quad (20)$$

$$h_p[J] : \{\{L_0\}, \{R_1\}, \{R_2\}\}^n \longrightarrow \{\{L_0\}, \{R_1\}, \{R_2\}\}$$

where, $h_p[J]$ is the J-type CTVL subfunction which has following AND or OR subfunctions.

$$g_4 \ [AND \ or \ OR] : \{R_2, R_1\} \longrightarrow \{R_2, R_1\}$$

$$h_{s1}[AND \ or \ OR] : \{\{R_1\}, \{L_o\}\} \longrightarrow \{\{R_1\}, \{L_o\}\} \quad (21)$$

$$h_{s2}[AND \ or \ OR] : \{\{L_o\}, \{R_2\}\} \longrightarrow \{\{L_o\}, \{R_2\}\}$$

These functions correspond to the ones in equation (14) exchanging the variable 1 by R_2, 0 by R_1 and R by L_o .

Example 8 [13,15]

The J-type 4 valued system which has 2 valued functional completeness can be constructed with the elements specified in Table 5.

Table 5   J-type CMVL (4 valued) element

Truth Table					Ordered Graph		Note
	0	1	R_1	R_2	R_1	R_2	b : 1 or 0
0	1	0	b	R_2	R_1 R_2		subdivided by
1	0	0	b	R_2			$f(1,R_1)= 1 \ or \ 0$
R_1	b	b	R_1	R_1	$\{1,0\}$		$f(0,R_1)= 1 \ or \ 0$
R_2	R_2	R_2	R_1	R_2	$\{1,0\}$		

Property 6

All stuck at $v_1, v_2, ---v_{r-2}, R_1$ and R_2 single and multiple faults are detected by applying the triple universal test $\mathbf{V}, \mathbf{R}_1$, and $\mathbf{R}_2$. If the output is in L_o for input $\mathbf{V}$, R_1 for $\mathbf{R}_1$ and R_2 for $\mathbf{R}_2$, the system is good, if otherwise the system is fault.

(2) Notes on applications

(a) Selection of CMVL functions

Fail safe system is constructed by,
i) letting the elements asymmetrically fault, (direction 1)
ii) selecting monotonically increasing functions, (direction 2) and matching these directions. As the common safety side, the redundant value is introduced, then the functions are restricted to C-type and Φ-type [6-10].

The equivalent Φ-type system is realized by 2 rail binaly system, and assignment of logical values sets are closely related to the code, these studies have been developed to the area of self-checking circuit design.

Applicable CMVL functions for testable design are classified, 1) A-type and C-type with one redundant logical value, 2) D-type and J-type with two redundant values, and they are extendable to arbitrary multiple-valued system.

(b) Elements implementation and fault modes

Various implementation proposales for fail safe logical elements have been done and at present Φ-

type is easily realizable [21,23]. The A-type is also realized with simple circuit (2-4 gates) [12], D-type has attractive property, but necessary number of gates increases (10 or more). The J-type has interesting property and it has applicability for special use, because it is realizable as a logic of three phase pulse signals [13].

Another problem is in multiplexity of fault modes of elements. Some of them may be not easily detectable and these modes depend on the elements circuit configurations [15,26]. For overcoming these problems, function transform or control strategies are proposed, but the fault coverage problem between normal operation state and controlled detectable state, is remained [24,25].

V. CONCLUSIONS

A unified classification, construction methods and properties of CMVL systems with monotonic subfunctions have been presented.

The concept of composite or hybrid logical function acquires a significance when the system is multiple-valued logic, because the MVL can have special and useful 2 valued and 3 valued subfunctions, but the subfunctions of 2 valued system are degenerated to the constants. These fact can be an origin of the important and interesting features of MVL.

In some CMVL discussed in this paper, the D-type and J-type are especially interesting. They have two or three monotonic subfunctions, AND or OR, and all stuck at constant faults are detectable with two or three universal (function independent) tests. They are probably most testable ones in all MVL. These CMVL with useful subfunctions will give a suggestion to a testable design of digital systems and other applications.

There remain following problems:
1) Studies on the CMVL's which have other special subfunctions other than the monotonic ones.
2) Overcoming the gate implementation problems which are common difficulties and obstacles in the applications of MVL.

VI. ACKNOWLEDGEMENT

This paper is a development of the studies on fail safe logical systems, the authors greately appreciate to the authers of referred papers and others, and also would like to thank to Mr. Shukuri Ryo for his cooperation in this reserch.

Thanks are due to referees for revising paper especially Definition 1.

REFERENCES

[1] D.C.Rine Ed., Computer Science and Multiple-Valued Logic —— Theory and Applications, North-Holland, 1977
[2] T.Hasegawa et al., Multiple-Valued Logics and Their Applications, II,III, Kyoto University, 1972, 1982
[3] K.C.Smith, Circuits for Multiple Logic —— A Tutorial and Application, Proc. of 7th Inter. Symp. on MVL , pp.30-43, May 1976
[4] S.M.Lee, Modern Switching Theory and Digital Design, Prentice-Hall, 1978

[5] H.Mine and Y.Koga, Basic Properties and a Construction Method for Fail Safe Logical Systems, IEEE Trans. Comput. EC-18, 3, pp.282-289, June 1967

[6] T.Takaoka and H.Mine, N-Fail-Saife Logical Systems, IEEE Trans. Comput., C-20, 5, pp.536-542, May 1971

[7] N.Tokura, T.Kasami and A.Hashimoto, Fail Safe Logic Nets, IEEE Trns. omput., C-20, 4, pp. 323-330, March 1971

[8] H.Hirayama, T.Watanabe and Y.Urano, Synthesis of Fail Safe Logical Systems, Trans. IECE Japan, 52-C, 1, pp.33-40, Jan. 1969

[9] M.Mukaidono, On the Mathematical Structure of the C-type Fail Safe Logic, Trans. IECE Japan, 52-C, 12, pp.812-819, Dec. 1969

[10] M.Mukaidomo, On the B-Ternary Logical Function - A Ternary Logic Considering Ambiguity, Trans. IECE Japan, 55-D, 6, pp.355-362, June 1972

[11] Y.Koga, Diagnosis of Fail Saif Logic System, Trans. IECE Japan, 58-D, 5, pp.264-270, May 1975

[12] M.Nakamichi and T.Yagi, Methods of Simplifying Fault Detection by Letting Monoterm Degneracy be Compatible with Functional Completeness, Trns. IECE Japan, 57-D, 9, pp.519-526, Sept. 1974

[13] M.Nakamichi, S.Ryo and H.Itoh, J-type 3 Valued Logical Systems and Their Fault-Detectability, Trans. IECE Japan, 59-D, 12 pp.941-943, Dec. 1976

[14] M.Nakamichi, S.Ryo and H.Itoh, Classification of Some 3 Valued Logics Using Ordered Graphs, Trans. IECE Japan, 59-D, 12, pp.943-944, Dec. 1976

[15] M.Nakamichi, H.Itoh and S.Ryo, Some Testable 3-Valued Logics and Their Applications, Paper of Technical Group on Reliability, IECE Japan, R78-2, pp.9-18, May 1978

[16] A.Druzeta and Z.G.Vranesic, A Higher Radix Technique for Fault Detection in Many-Valued Multithreshold Networks, IEEE Trans. Comput., C-27, 11, pp.1070-1073, Nov. 1978

[17] R.Betancourt, Derivation of Minimum Test Set for Unate Logical Circuits, IEEE Trns. Comput. C-20, 11, pp.1264-1269, Nov. 1971

[18] S.B.Akers JR, Universal Test Sets for Logic Networks, IEEE Trns. Comput. C-22, 9, pp.1016-1020, Nov. 1973

[19] S.M.Reddy, Complete Test Sets for Logic Functions, IEEE Trans. Comput. C-22, 9, pp.1016-1020, Nov. 1973

[20] J.P.Hayes, On Modifying Logic Networks to Improve Their Diagnosability, IEEE Trans. Comput., C-23, 1, pp.56-62, Jan. 1974

[21] M.Nakamichi and N.Fujikura, Composite Construction Methods of Three Valued and Leveled Logical Elements and Their Classification, Trans. IECE Jap-n, 57-D, 11, pp.605-612, Nov. 1974

[22] M.Nakamichi, Fail Safe Logic and Their Application in Contactless Relay Systems, Control Engineering, Japan 13, pp.116-128, Feb. 1969

[23] S.Tsuchiya, On Ternary C-type Fail-Saif Logic Circuits, Jour. of SICE, Japan, 6-1, pp.81-88, Jan. 1970

[24] M.Sakauchi and H.Inose, Synthesis of Fault Diagnosable Logical Circuits by Function Conversion Method, Trans. IECE Japan, 56-D, 1, pp.47-54, Jan. 1973

[25] M.Nakamichi and K.Takemura, Simplification of Fault Detection Utilizing the Power Circuit Lines Control of Composite Type 3 Valued Systems, Trans. IECE Japan, 58-D, 3, pp.113-120, March 1975

[26] M.Hu and K.C.Smith, On the Use of CMOS Ternary Gates to Realize a Self-Checking Logic System, Proc. of 11th Inter. Symp. on MVL, pp.212-215, May 1981

APPENDIX

Notes on subfunctions and fail safe property

(1) Subfunctions and definition

A class of MVL which contains $\ddagger$-type and C-type A-type as special ones can be defined as the CTVL with subfunctions specified in equation (6). In this case, subfunctic. h_p has monotonically increasing property for R, but $\{1,0\}$ can not always be treated as a single equivalent value.

For example, the CTVL defined above are realized with the elements specified in Table 6. From Table 6, it is seen that the output for input combination R and $\{1,0\}$, separates into R and $\{1,0\}$.

Table 6 Φ-type and J-type NAND

Φ-type				J-type			
Truth Table			OrderedGraph	Truth table			OrderedGraph
\	0	1	R	1 0	\	0	1	R	1 0
0	1	1	1		0	1	1	R	
1	1	0	R		1	1	0	0	
R	1	R	R		R	R	0	R	

(2) Fail safe property

The $\ddagger$-type system which is realized with the $\ddagger$-type elements has the fail safe property, but the system realized with J-type elements has not. The difference comes from the difference of properties of each NAND element subfunction shown as follows:

$$\begin{aligned} \Phi &: f(R,0)=f(1,0)=f(0,0) \quad f(R,1)=f(R,R)=R \\ J &: f(R,1)=f(1,1){\neq}f(0,1) \quad f(R,0)=f(R,R)=R \end{aligned} \quad (22)$$

The difference is also indicated in ordered graph in simple form :

(i) In Φ-type, the node of sensitizing logical value (1 in NAND, 0 in NOR) has two exit lines to nodes R and non-sensitizing or advantageous one (0 in NAND, 1 in NOR). This is necessary condition for fail safe logical elements. and this condition is also satisfied in C-type elements in Fig. 1, but not satisfied in J-type and A-type.

(ii) There exists most advantage value in $\{1,0\}$ and the node of this value also has two entry lines which bring about information loss less property in Φ-type, but C-type has not [5-10].

(iii) It is also known that test procedure can be simplified using monotonic properties [17-19]. If the faults are all at R side, they can be detected by the output R, and the diagnosability is also improved [11].

AN ALGEBRAIC METHOD FOR HAZARD ANALYSIS WITH THE MAXIMUM NUMBER OF SPIKES IN COMBINATIONAL AND SEQUENTIAL CIRCUITS

by Janusz Rajski and Jerzy Tyszer

Regional Computer Center, Technical University
of Poznań, Poznań, Poland

Abstract

A method is described for hazard a-
nalysis with the maximum possible number
of spikes in combinational and sequential
circuits with NAND gates and flip-flops.
This paper studies the conditions of cre-
ation and propagation of hazards. The in-
fluence of hazards of inputs on steady-
state of the outputs of a flip-flop with
synchronous and asynchronous triggering is
considered. Partial results are given in
A4 alphabet, where we assume all lines to
be 0 or 1. Final, complete results are
given in A11 alphabet, where some lines
may be x, i.e. in unknown state.

Introduction

The problem of hazard analysis has
been extensively studied in literature.
Several methods of hazard analysis have
been proposed for variety of applications.
One application is the design of hazard-
free combinational circuits. Most of the
early papers are limited to particular
classes of hazards such as hazards of a
single input variable, static hazards [3]
or dynamic hazards [6]. Thayse and Davio
[5] introduced separate criteria for
static, dynamic and sequential hazards
using Boolean differential calculus. Uni-
fied approach to combinational hazards was
presented by Meister [1]. The approach is
based on a pure delay model of combina-
tional circuits and defines a unatness
criterion for the appearence of static and
dynamic hazards.

Breuer and Harrison [2] introduced a
method of hazard analysis based on hazard
status flags. The hazard status of a line
can be: hazard free (hf), hazard present
(hp) or hazard status unknown (hsu). The
problem of creation of hazards, creation
of hazard free requirements and propaga-
tion of hazard status flags was considered
for several circuit element types and for
the limited number of stimuli, e.g. the
conditions for creation of a hazard are
formulated as a collection of input stimu-
li. Thus, the approach as not general may
lead to difficulties when applied to a
wide spectrum of circuit elements.

All the abovementioned methods do not
investigate the number of spikes that may
occur during a hazardous transition. In
sequential circuits the hazard analysis
based on the hazard status flag may lead
to pessimistic results, especially in the
case of counters and shift registers. The
approach introducing the maximum possible
number of spikes during hazardous transi-
tion better estimates the range of hazard
influence in sequential circuits.

The hazard analysis with the maximum
number of spikes in combinational circuits
was introduced by Zisapel et al. [7]. This
approach uses the extended sum of product
representation of a combinational circuit
introduced by Unger [6]. The method is
functional in nature and can not be easily
implemented in logic simulator.

The approach presented in this paper
is rather structural than functional and
may be easily implemented in a logic table
driven simulator with selective trace. The
method is modular since we do not assume
that an input variable changes only once
during a single transition.

Basic notions and assumptions

In this paper we consider circuits
composed of NAND gates and flip-flops con-
nected without feedback. For the analysis
of steady states and hazards in sequential
circuits we introduce the notion of gen-
eral flip-flop. It satisfies the following
conditions:
1. Asynchronous inputs, denoted respective-
ly by S and R, set the flip-flop on 1
or 0,
2. Synchronous inputs, denoted by J and K,
set the flip-flop on 1 or 0 if asynchro-
nous inputs are not active,
3. Clock input T may be regarded as edge or
level triggered.

In order to avoid the duality of consider-
ation, we assume that the direct set S and
reset R input lines are active when low
and S is dominant. It does not lead to

loss of generality. Similarly we assume that $r\epsilon\{\bar{z},z\}$, where z denote write signal.

The problem of circuit delays is not studied extensively. It is assumed that basic elements introduce unbounded but finite delays. The changes of the inputs of a circuit may but need not take place simultaneously. We do not restrict the number of changes on any line, even if it is an input.

Definition 1: A line of a circuit contains a hazard if the number of changes between two steady states during a transition is greater than one.

Since a hazard occurs between two steady-state line values we shall usually represent the value of a line as a triple $S_a^- S_a^+ L_a$, where S_a^- denotes the state of a line a before a transition, S_a^+ denotes the state of a line a after the transition and L_a is the maximum number of spikes during the transition. In the systems C4 and S4 we assume that S_a^-, $S_a^+ \epsilon \{0,1\}$ for any line a of a circuit. In the systems C11 and S11 the set of steady states is extended and $S_a^- \epsilon \{0,1,x\}$, $S_a^+ \epsilon \{0,1,x,\bar{x},x^x\}$, where $x,\bar{x}$ and x^x represent line values not yet specified to be either 0 or 1.

A transition of a line is the transient process which is initiated by the value S_a^- and ends when the line reaches the value S_a^+. Examples of the extended description of a line value and corresponding transitions are shown in Fig.1.

S^-	S^+	L	Steady states	Transitions
0	0	0		
0	0	2		
1	0	0		
1	0	3		

Figure 1

Transition of a line may be described as a sequence S_a of 0 and 1, where the first and last element correspond to the steady-state values. Let us define three variables d_{ab}^{0x}, d_{ab}^{x0} and d_a^1 as follows:

$$d_{ab}^{0x} = \begin{cases} 1, & \text{if } S_a \text{ and } S_b \text{ begin with 0,} \\ 0, & \text{otherwise.} \end{cases}$$

$$d_{ab}^{x0} = \begin{cases} 1, & \text{if } S_a \text{ and } S_b \text{ end with 0,} \\ 0, & \text{otherwise.} \end{cases}$$

$$d_a^1 = \begin{cases} 1, & \text{if } S_a \text{ comprises 1,} \\ 0, & \text{otherwise.} \end{cases}$$

Lemma 1: The maximum number of changes at the output w of the NAND gate with inputs a and b is

$$c_w = d_a^1 d_b^1 (c_a + c_b - d_{ab}^{0x} - d_{ab}^{x0}),$$

where c_a and c_b denote the number of changes in the sequences S_a and S_b.

The proof of this lemma can be found in [4].

Hazard analysis for known line values

In this section we shall consider the system C4 for hazard analysis in combinational circuits and the system S4 for hazard analysis in flip-flops. For this case we assume that $S_a^-, S_a^+ \epsilon \{0,1\}$. The value of a line a can be briefly denoted as $D_a L_a$, where $D_a \epsilon \{00,01,10,11\}$.

Theorem 1: The extended value of the output w of the NAND gate with input values $D_a L_a$ and $D_b L_b$ is $D_w L_w$. D_w is given by Table I and L_w is determined by Table II.

$$>\underset{-}{\vee}1 \quad L_w = d_a^1 d_b^1 (L_a + L_b - 1)$$

$$>+ \quad L_w = d_b^1 (L_a + L_b)$$

$$\underset{+}{\vee} \quad L_w = d_a^1 (L_a + L_b)$$

$$+1 \quad L_w = L_a + L_b + 1$$

$$+ \quad L_w = L_a + L_b$$

Table I

D_b\\D_a	00	01	10	11
00	11	11	11	11
01	11	10	11	10
10	11	11	01	01
11	11	10	01	00

Table II

D_b\\D_a	00	01	10	11
00	$>\underset{-}{\vee}1$	$>+$	$>+$	$>+$
01	$\underset{+}{\vee}$	$+$	$+1$	$+$
10	$\underset{+}{\vee}$	$+1$	$+$	$+$
11	$\underset{+}{\vee}$	$+$	$+$	$+$

Proof: Table I is due to Boolean algebra when applied to initial and final steady-state of transition. The number of spikes is limited by the number of changes according to $L_a = c_a/2$. Since each spike is composed of two changes and additionally for $D_a = 01,10$ there is one more change introduced by steady-states the relation between the number of changes and the number of spikes for various steady states is as it is shown in Table III. Based on Lem-

307

ma 1 and Table III the maximum number of changes for all elements in Table II can be determined as for element $(1,1)$:

$$c_{11} = d_a^1 d_b^1 (2L_a + 2L_b - 1 - 1) = d_a^1 d_b^1 2(L_a + L_b - 1)$$

and the maximum number of spikes is

$$L_{11} = c_{11}/2 = d_a^1 d_b^1 (L_a + L_b - 1)$$

This calculation has to be performed for all elements in Table II.　　　Q.E.D.

Corollary: a/ coordinates $(2,3)$ and $(3,2)$ describe the conditions for creation of hazard, b/ coordinates $(2,2),(3,3),(4,4),(2,4)$ $(3,4),(4,3)$ and $(4,3)$ describe unconditional propagation of a hazard from both inputs a and b, c/ coordinates $(1,1-4)$ describe conditional propagation of a hazard under the condition $L_b > 0$, d/ coordinates $(1-4,1)$ describe conditional propagation of a hazard under the condition $L_a > 0$.

　　In the analysis of a flip-flop the following sequence of this analysis is assumed:
1. The calculation of output states from asynchronous inputs.
2. The calculation of output states from synchronous inputs, if step 1 does not definitely determine the state of output.

　　We introduce a function ff, which determines steady-state values of a flip-flop output.

Definition 2 : Let ff be a function

$$ff: S_S^- S_S^+ \times S_R^- S_R^+ \times S_Q^- \rightarrow S_Q^- S_Q^+$$

defined by Table IV, where z describes influence of synchronous inputs on steady-state of an output, S describes relation between synchronous inputs.

Table IV

R\S	00	01	10	11
00	11	10	01	00
01	11	1X	01	$0z(0,X)$
10	11	10	S_Q^-1	S_Q^-0
11	11	$1z(1,X)$	S_Q^-1	$S_Q^- z(S_Q^-,S)$

　　Note that the steady-state of an output may be affected by synchronous inputs if $S_S^+ = S_R^+ = 1$. However only in the case $S_S^- S_S^+ = S_R^- S_R^+ = 11$ this influence is uniquely determined. Otherwise simultaneous changes on synchronous and asynchronous inputs lead to sequential hazard, i.e. unknown state of an output. This is why the output has been extended by the symbol x.

Definition 3: Let A and B be states. The operation ✳ is defined as follows:

$$A \mathbin{\textstyle *} B = \begin{cases} A, & \text{if } A = B, \\ x, & \text{if } A \neq B. \end{cases}$$

Let (T, R_T, R_A) denotes write conditions from synchronous inputs, where $T \in \{\bar{z}, z\}$, R_T, R_a denote hazard flags on input T and asynchronous inputs respectively, R_T, $R_A \in \{0,1\}$ and $R_A = R_S + R_R$.

Definition 4: Let O and I be an initial state of a flip-flop output and a state of a flip-flop output, if the asynchronous inputs are not active and the clock input is active. The value of $z(Q,I)$ is given in Table V.

Table III

D_a	c_a
00	$2L_a$
01	$2L_a + 1$
10	$2L_a + 1$
11	$2L_a$

Table V

$R_T R_A$ / T	00	10	-1
$\bar{z}$	Q	Q✳I	X
z	I	I / X*	X

*for T-type flip-flop

　　Relations between synchronous inputs and their influence on the steady state of a flip-flop are described by function S.

Definition 5: The steady-state of a flip-flop output as a function S of their synchronous inputs and hazard status flag is defined by Table VI.

Table VI

$R_J R_K$ / $J K$	00 00	00 11	11 00	11 11	-- --
0	$S_Q^- S_Q^-$	$S_Q^- 0$	$S_Q^- 1$	$S_Q^- \bar{S}_Q^-$	$S_Q^- X$
1	$S_Q^- X$	$S_Q^- X$	$S_Q^- X$	$S_Q^- X$	$S_Q^- X$

Table VI similary like in the case of function ff, is not closed on the set A4. In all cases where one of synchronous inputs changes its steady-state the output of a flip-flop is unknown.

　　Now we shall determine the maximum possible number of spikes on the output of a flip-flop. Let us untroduce following symbols:

$$d_a^s = \begin{cases} 1, & \text{if } S_a \text{ contains a state different from s,} \\ 0, & \text{otherwise.} \end{cases}$$

$$I = \begin{cases} 1, & \text{if } S_R^- \neq 0 \wedge S_S^- \neq 0 \wedge S_S^+ \neq 1 \wedge S_Q^- \neq 0, \\ 0, & \text{otherwise.} \end{cases}$$

$$D = \begin{cases} 1, & \text{if } S_S^- \neq 0 \wedge S_R^- \neq 0 \wedge S_Q^- \neq 1 \wedge d_S^1 = 1, \\ 0, & \text{otherwise.} \end{cases}$$

　　The transition process of a flip-flop will be regarded as a sequence of its

states grouped in three phases: A_I, N, A_{II} delimited by four states: S_Q^-, B_Q^-, B_Q^+, S_Q^+ /Fig.2/. It is assumed that with each point in Fig.2 there are connected 6 momentary values: S,R,J,K,T,Q. Two neighbouring points may differ at one position

Figure 2

at least. In some cases a phase is even the whole transition may be reduced to one single point.

S_Q^- and S_Q^+ represent the initial and final steady-state. During phase N the condition $S,R = 1$ is fulfilled by coincidence of non active steady or momentary states in a transition. In phase A_I and A_{II} the case $S,R = 1$ may occur but it is not necessary.

Theorem 2: The maximum number of spikes L_Q observed on the output of a flip-flop is

$$L_Q = \lfloor (c_Q^a + c_Q^{syn} d_S^0 d_R^0)/2 \rfloor$$

where $c_Q^a = (c_S - D + I)d_R^1 + D$ is the maximum possible number of changes on the output caused by asynchronous inputs, c_Q^{syn} is the maximum possible number of changes on the output caused by synchronous inputs.

Proof: The transition process of a flip-flop may be represented as a directed graph /Fig.3/, where vertices are described by triples (S,R,Q), $S,R,Q \in \{0,1\}$ and edges represent changes. The nature
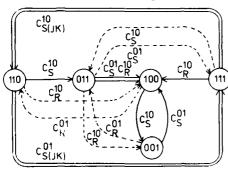
Figure 3

of the graph is that every transition from one vertex to another is equivalent to changes observed on the output /solid lines/ or unobserved /dashed lines/. The transition process between the initial steady-state (S_S^-,S_R^-,S_Q^-) and final steady-state (S_S^+,S_R^+,S_Q^+) may be considered as a

path in the graph between two vertices. The maximum possible number of changes on the output is equal to the maximum length of the path in the graph. If the initial vertex is (100) or (001), all changes on input S are observed on the output. In this case $d_R^1 = 1$, $D = 0$ and $I = 0$. When starting from vertex (110) or (011), changes on input S are observed on the condition that input R changes its state $d_R^1 = 1$, $I = 0$, then $c_Q^a = c_S - D + D + I = c_S$. In the other case only single change is observed C_S^{10} if vertex (110) is the initial vertex. In this case $D=1$ and $c_Q^a = (c_S - 1 + 0)0 + 1 = 1$. If the transition starts from (111), the change C_R^{10} is observed and additionally it enables changes of S to affect the output. Here we have $I = 1$. If $d_R^1 = 1$, then $c_Q^a = (c_S - 0 + 1)1 + 0 = c_S + 1$, if $d_R^1 = 0$ /no change on R/ then $c_Q^a = 0$.

Q.E.D.

It may be shown in the graph 4 that the maximum number of output changes $c_Q^{a'}$ caused by asynchronous inputs with the interference of synchronous inputs is $c_Q^{a'} = c_Q^a + \Delta$, where the variable Δ is defined by Table VII. The rows in the table correspond to the odd and even number of output changes caused by synchronous inputs. For

Table VII

The number of changes $\diagdown$ $B_Q^- B_Q^+$	00	01	10	11
odd	1	-1	-1	-1
even	0	0	0	-2

given sequences S_S and S_R the values of B_Q^- and B_Q^+ can be calculated from the graph by checking whether $B_Q^- B_Q^+$ reaches the values 00,01 or 10 and 11 [4].

The sequence of output state caused by synchronous input will be denoted as $S_{S(JK)}$. For given sequences S_S, S_R and $S_{S(JK)}$ variable τ will be defined as follows:

$$\tau = \begin{cases} 1, & \text{if the first element of } S_{S(JK)} \text{ is different from } B_Q^-, \\ 0, & \text{otherwise.} \end{cases}$$

Note that if one of asynchronous in-

Table VIII

D_b \ D_a	00	0X	01	X0	XX	XX^X	X$\bar{X}$	X1	10	1X	11
00	11 / V·1	11	11	11	11	11	11	11	11	11	11
0X	V +	1X / +	1X	11	1X	1X	1X	1X	11	1X	1X
01	V +	+	10 / +	11	1X	1X	1X	10	11	1X	10
X0	V +	+1	+1	X1 / +	X1	X1	X1	X1	X1	X1	X1
. XX	V +	+	+	+	XX / +	XX^X	XX^X	XX^X	X1	XX^X	XX
XX^X	V +	+	+1	+1	+	XX^X / +1	XX^X	XX^X	X1	XX^X	XX^X
X$\bar{X}$	V +	+	+1	+1	+	+1	XX^X / +1	XX^X	X1	XX^X	X$\bar{X}$
X1	V +	+	+	+1	+	+1	+1	X0 / +	X1	XX^X	X1
10	V +	+1	. +1	+	+	+1	+1	+1	01 / +	01	01
1X	V +	+1	+1	+	+	+1	+1	+1	+	0X / +	0X
11	V +	+	+	+	+	+	+	+	+	+	00 / +

310

puts remains in non active state there are changes on the other asynchronous input and $\tau=1$, then all changes are observed on an output if synchronous inputs are active. Let $ds=c_S^{10}-c_S^{01}$ and $dr=c_R^{10}-c_R^{01}$ and ca_T denote active state on clock input. It may be shown [4], that the number of changes c_Q^{ase} on an output caused by the interference of changes on one of the asynchronous inputs and the clock input is

$$c_Q^{ase}= T+(T-ds\ \overline{d}_R^1-dr\ \overline{d}_S^1)(\overline{d}_S^1+\overline{d}_R^1)$$

where

$$T=\min\ (ca_T,\tau d_S^1 d_R^1+c_S^{10}\overline{d}_R^1 d_{S(JK)}^1+c_R^{10}\overline{d}_S^1 d_{S(JK)}^0)$$

Now we define an operation $\overline{\min}$ over variables c_J and c_K:

$$\overline{\min}(c_J,c_K)=\begin{cases}\min(c_J,c_K)+1, & \text{if }S_J^-=S_K^-=0\text{ and}\\ & B_Q^-\text{ is different from the}\\ & \text{state after the first}\\ & \text{change on inputs J and K,}\\ \min(c_J,c_K), & \text{otherwise.}\end{cases}$$

The maximum number of changes observed on the output of a flip-flop generated by synchronous inputs for the edge-triggering ce_Q^{syn} and level-triggering cl_Q^{syn} flip-flop is respectively:

$$ce_Q^{syn}=(\min(\overline{\min}(c_J,c_K),ca_T-T)+\Delta+c_Q^{ase})\overline{e}_{JK}^{11}$$

$$+(ca_T+\Delta+c_Q^{ase}-T)e_{JK}^{11},$$

$$cl_Q^{syn}=(\overline{\min}(c_J,c_K)+\Delta+c_Q^{asl})\overline{e}_{JK}^{11}d_T^n,$$

where

$$c_Q^{asl}=\tau d_S^1 d_R^1+2c_S^{10}\overline{d}_R^1 d_{S(JK)}^1+2c_R^{10}\overline{d}_S^1 d_{S(JK)}^0$$
$$-drd_S^1-dsd_R^1$$

$$e_{JK}^{11}=\begin{cases}1, & \text{if it is possible to take value}\\ & J=K=1\text{ at the same time.}\\ 0, & \text{otherwise.}\end{cases}$$

Note that the second component of the ce_Q^{syn} /describing T-type flip-flop/ is a special case of the first part. Since $\overline{\min}(c_J,c_K)=ca_T$, then $\min(ca_T,ca_T-T)+\Delta+c_Q^{ase}$ $=ca_T-T+\Delta+c_Q^{ase}$.

Smilary for cl_Q^{syn} each change of data on synchronous input is observed on an output if the clock input is in active state. Therefore the analysis of changes in this flip-flop may be performed in an edge-triggering flip-flop with the assumption that the number of clock changes is sufficiently great, e.g. $ca_T=\infty$.

Hazard analysis for unknown values

The main defficiency of the system S4 is that it is not closed to the set A4. This is because of well known property of

switching circuits which says that: a hazard in combinational circuit may set a flip-flop to an unknown value. Some lines may be set to an unknown value during power-up or initialization. When a circuit is initially power-up, the initial state of flip-flops is unpredictable.

In the alphabet A11 used in this section we introduce the value x which denotes unknown state. Symbol xx denotes that before and after a transition the value of a line is unknown but the same; xx^x denotes that it is not known whether the unknown value after a transition is the same as before a transition; $x\overline{x}$ denotes that the value is unknown but opposite.

In this section we consider system C11 for hazard analysis in combinational circuits composed of NAND gates and system S11 for circuits with flip-flops. Line states take values from the alphabet A11; $A11 = \{00,0x,01,x0,xx,xx^x,x\overline{x},x1,10,1x,11\}$.

Theorem 3: The extended value of the output w of a NAND gate with input values $D_a L_a$ and $D_b L_b$ is $D_w L_w$. D_w and L_w is given by Table VIII. D_w is given in table above the main diagonal, L_w is determined below the main diagonal in the same table.

Proof: The upper part of Table VIII, above the main diagonal is due to the behaviour of a NAND gate in three-valued logic. The lower part describing the maximum number of spikes is by Theorem 1, because x denotes unknown steady state 0 or 1. It is evident that elements in intersections of rows and columns 1,3,9 and 11 in Table VIII correspond to intersections of rows and columns 1,2,3 and 4 in Table II. In all remaining cases where symbol x is encountered, we have to check value 0 and 1 and choose the case which leads to the greater number of spikes. If we denote the number of spikes on the output of a gate by $1(D_a L_a, D_b L_b)$, then for instance for row 2 we have:

$$1(0xL_a, D_b L_b)=\max(1(00L_a, D_b L_b),1(01L_a, D_b L_b))$$
$$=1(01L_a, D_b L_b).$$

Full mapping is given in Table IX.

Table IX

Table VIII	1	2	3	4	5	6	7	8	9	10	11
Table II	1	2	2	3	4	2,3	2,3	2	3	3	4

Single element in Table VIII corresponds to 1,2 or 4 elements in Table II. Then the function is selected, which gives the greater number of spikes according to the order:

$$>^V_1, >+, +, +1.$$

311

Table X

S\R	00	0X	01	X0	XX	XX^X	X$\bar{X}$	X1	10	1X	11
00	11	1X	10	X1	XX	XX^X	X$\bar{X}$	X0	01	0X	00
0X	11	1X	1X	X1	Xz(X,X)	XX^X	XX^X	XX^X	01	0X	0z(0,X)
01	11	1X	1X	X1	Xz(X,X)	XX^X	XX^X	XX^X	01	0X	0z(0,X)
X0	11	1X	10	X1	XX^X	XX^X	XX^X	X0	01 X1 –	0X XX^X –	00 X0 –
XX	11	1X	1X	X1	Xz(X,X)	XX^X	XX^X	XX^X	01 X1 –	0X XX^X –	0z(0,X) Xz(X,X) –
XX^X	11	1X	1X	X1	XX^X	XX^X	XX^X	XX^X	01 X1 –	0X XX^X –	0z(0,X) Xz(X,X) –
X$\bar{X}$	11	1X	1X	X1	XX^X	XX^X	XX^X	XX^X	01 X1 –	0X XX^X –	0z(0,X) Xz(X,X) –
X1	11	1X	1X	X1	XX^X	XX^X	XX^X	XX^X	01 X1 –	0X XX^X –	0z(0,X) Xz(X,X) –
10	11	1X	10	– X1 11	XX^X	– XX^X 1X	– XX^X 1X	– X0 10	01 X1 11	0X XX 1X	00 X0 10
1X	11	1X	1X	– X1 11	XX^X	– XX^X 1X	– XX^X 1X	– XX^X 1X	01 X1 11	0X XX 1X	0z(0,X) XX^X 1X
11	11	1z 1,X	1z 1,X	– X1 11	– Xz(X,X) 1z(1,X)	– XX^X 1z(1,X)	– Xz(X,X) 1z(1,X)	– Xz(X,X) 1z(1,X)	01 X1 11	0X XX 1z(1,X)	0z(0,S) Xz(X,S) 1z(1,S)

$S_Q^- = 0$
$S_Q^- = X$
$S_Q^- = 1$

312

The table received in this way is equivalent to Table VIII.

<div align="right">Q.E.D.</div>

Similary we extend system S4 to S11 to describe steady states and hazards in A11 alphabet.

Definition 6: Let ff be a function

$$ff : S_S^- S_S^+ \times S_R^- S_R^+ \times S_Q^- \rightarrow S_Q^- S_Q^+$$

defined by Table X.

The operation $*$ will be newly defined in alphabet A11.

Definition 7: Let A and B be states in alphabet A11. The operation $*$ is defined as follows

$$A * B = \begin{cases} A, & \text{if } A=B \text{ and } A,B \neq x, \\ x, & \text{if } A \neq B, \\ x^X, & \text{if } A=B=x. \end{cases}$$

As a result of unknown states in alphabet A11 the alphabet of write conditions has been extended from $\{\bar{z},z\}$ to $\{z, z, z^X\}$, where z^X denotes a possible write signal. The operator z is extended as it is shown in Table XI.

Table XI

T \ $R_T R_A$	00	10	-1
$\bar{z}$	Q	Q$*$I	Q$*$X
z	I	I	Q$*$X
$z^\bullet$		Q$*$I	
z^X	Q$*$I	Q$*$I	Q$*$X

$^\bullet$for T-type flip-flop

Example 2: Let us consider 4-bit binary reverse counter in the initial state $S_{Q1}^-=0$, $S_{Q2}^-=1$, $S_{Q3}^-=0$, $S_{Q4}^-=1$ which is clocked by a simple combinational circuit. The clock input is sensitized by state $S_T^- S_T^+ L_T = 00\text{-}2$ /based on Theorem 1/. The state of the counter is determined in following calculations:

Q1: $S_Q^+ = z(S_Q^-,S_Q^-)= x$, because $R_T=1$ and $T=\bar{z}$

$L_Q = \lfloor (\lceil ca_T + + c_Q^{ase} - T)/2 \rfloor$

$= \lfloor (2 + 0 + 0 - 0)/2 \rfloor = 1$,

Q2: $S_Q^+ = z(S_Q^-,S_Q^-)= x$, because $R_T=1$, $T=z^X$,

$L_Q = \lfloor (1 + 0 + 0 - 0)/2 \rfloor = 0$,

Q3: $S_Q^+ = z(S_Q^-,S_Q^-)= S_Q^- = 0$, because $R_T = 0$ and $T=\bar{z}$,

$L_Q = \lfloor (0 + 0 + 0 - 0)/2 \rfloor = 0$,

Q4: $S_Q^+ = z(S_Q^-,S_Q^-)= S_Q^- = 1$, because $R_T = 0$ and $T=\bar{z}$,

$L_Q = 0$.
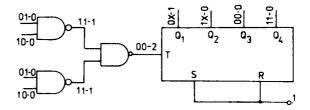
Figure 4

Conclusions

This paper describes an algebraic method for hazard analysis with the maximum number of spikes. Basic primitives in this paper are NAND gates and flip-flops. The method can be easily programmed for automatic processing. It is especially convenient for simulation analysis of digital circuits and it may be easily implemented in a table-driven simulator with selective trace. The existence of a hazard is not only detected but also the maximum number of spikes is calculated. The results are more accurate when dealing with such circuits like ripple counters or registers.

References

[1] J.Beister, "A unified approach to combinational hazards", IEEE Trans. Comp. vol. C-23, pp.566-575, June 1974.

[2] M.Breuer, R.L.Harrison, "Procedures for eliminating static and dynamic hazards in test generation", IEEE Trans. Comput., vol. C-23, pp.1069 - 1078, October 1974.

[3] E.B.Eichelberger, "Hazard detection in combinational and sequential switching circuits", IBM J. No 2, pp.90-99, 1965.

[4] J.Rajski, J.Tyszer, "Detection and propagation of hazards with the maximum possible number of spikes", submitted to IEEE Trans. Comput.

[5] A.Thayse, M.Davio, "Boolean differential calculus and its application to switching theory", IEEE Trans. Comput. vol. C-22, pp.409-419, April 1973.

[6] S.H.Unger, "Asynchronous Sequential Switching Circuits", New York :Wiley 1969.

[7] Y.Zisapel, M.Krieger, J.Kella, "Detection of hazards in combinational switching circuits", IEEE Trans. Comp. vol. C-27, pp.52-56, January 1979.

FOUR-VALUED LOGIC, STAR ALGORITHM AND THEIR APPLICATIONS

Ting-huai Chen, You-guang Yuan, Zhi-mo Liu, Zhi-min Zhang

Chongqing University, P.R. China

ABSTRACT----This paper is a survey of the works done in Chongqing University on Four-Valued Logic which is useful for describing the dynamic behaviors of logical objects. The mathematical Basis of Four-Valued Logic has been stated. The deduction method----Star Algorithm has been introduced. Three kinds of applications based on different explainations of the component have been dealt with, such as the fault detection of the combinational and the sequential circuits, the hazardous test generation as well as the transition logic. This paper is only an outline and the details should be referred to the original papers. See [1], [2], [3], [4], [5], [6].

0. INTRODUCTION

Two-valued logic is well known to everyone for it provides an efficient implement for studying the static behaviors of the objective phenomena. For example, these two values '1' and '0' can be used to represent the vapor state and the liquid state of water, high and low voltage levels in logic circuit, truth and falsehood of a proposition, etc.. But it is inconvenient to describe the dynamic behavior which exists obviously in the real world, for example, the condensation and evaporation of water, the fall and rise of a voltage pulse, the transition from state '1' to state '0' and the transition in the reverse order, etc.. In order to denote these transitional states we introduce two additional states D & $\overline{D}$ to the static states 1 and 0. Therefore we can extend two-valued logic to four-valued logic, which is useful for the description of the dynamic phenomena and contains the static logic as its special case.

In section I of this paper we state the mathematical basis of four-valued logic B_4 and introduce Star Algorithm, the deduction technique of this four components. Then we apply this theory to three different areas based upon different explainations of the real meanings of these four elements. Section II is the test generatin for fault detection in combinational and sequential circuits. Section III is the hazardous test generation for combinational circuit. Sec-

tion IV is the transition logic which is the generalization of the classical proposition logic. And in Section V we give a summary and list out some open problems.

I. MATHEMATICAL BASIS [1]

1.1 Four-Valued Boolean Algebra

Definition 1.1 Let B_2 be the Boolean algebra containing only two elements "1" and "0" and having three logical operations ".", "+","−" over them, then the direct product $B_2 * B_2$ is still a Boolean algebra denoted by B_4.

Thus if $x, y \in B_4$, then $x=(a_1, a_2)$, $y=(b_1, b_2)$ with $a_1, a_2, b_1, b_2 \in B_2$, and $\bar{x}=(\bar{a}_1, \bar{a}_2)$, $x \cdot y =(a_1 \cdot b_1, a_2 \cdot b_2)$, $x+y=(a_1+b_1, a_2+b_2)$. These operations ".", "+","−" in B_4 satisfy all the fundamental formulas of Boolean algebra. If $x=(a_1, a_2)$, then a_1 and a_2 are called the _front_ and _rear_ components of vector x.

The elements in B_4 are exactly
$$\theta = (0, 0), \quad D = (0, 1), \quad \bar{D} = (1, 0), \quad I = (1, 1).$$
and the operation tables are shown in Table 1.1.

x	$\bar{x}$	$+$	θ	I	D	$\bar{D}$	$\cdot$	θ	I	D	$\bar{D}$
θ	I	θ	θ	I	D	$\bar{D}$	θ	θ	θ	θ	θ
I	θ	I	I	I	I	I	I	θ	I	D	$\bar{D}$
D	$\bar{D}$	D	D	I	D	I	D	θ	D	D	θ
$\bar{D}$	D	$\bar{D}$	$\bar{D}$	I	I	$\bar{D}$	$\bar{D}$	θ	$\bar{D}$	θ	$\bar{D}$

Table 1.1 The operation table of B_4

1.2 Boolean Expression and Boolean Function

Definition 1.2

The Boolean expression of n variables is defined below

(1) o and 1 are Boolean expressions.

(2) The variables $x_1, x_2, \ldots, x_n$ are expressions.

(3) The Expression formed by using the operations ".", "+","−" finite times is also a Boolean expression.

(4) Only those expressions satisfying (1), (2), (3) are expressions.

Definition 1.3

Two Boolean expressions of n variables $F(x_1, x_2, \ldots, x_n)$, $G(x_1 x_2 \ldots, x_n)$ are equivalent denoted by $F(x_1, \ldots, x_n) \sim G(x_1, \ldots, x_n)$, if and only if one of them can be derived from the other by using the Boolean identities finite times.

Theorem 1.1

$F(x_1, \ldots, x_n) \sim G(x_1, \ldots, x_n)$ iff $F(x_1, \ldots, x_n)=G(x_1, \ldots, x_n)$ for all x_i's in B, where B is any Boolean algebra including B_2 and B_4

Definition 1.4

Let $F(x_1, \ldots, x_n)$ be a Boolean expression, when the values of all x_i's are taken from B_4, then $y=F(x_1, \ldots, x_n)$ defines a mapping $B_4^n \to B_4$, called a _Boolean Function_ corresponding to the expression $F(x_1, \ldots, x_n)$.

1.3 The Set of Mappings $B_4^n \to B_4$

Let $F(x_1, \ldots, x_n)$ be any mapping $B_4^n \to B_4$ and $\{F_n\}$ be the set of all these mappings $B_4^n \to B_4$, then $\{F_n\}$ contains 4^{4^n} functions, among them only 2^{2^n} functions are Boolean. Each F_n can be uniquely represented by a

table, a normalized vector form or the cano-
nical form using minimum terms. The set $\{F_n\}$
forms a Boolean algebra B_4^4 , it contains all
the 2^{2^n} Boolean functions, which can be re-
presented by Boolean expressions, as its
sub-Boolean-algebra. Here, the vector form
is defined by

Definition 1.5

If we associate a variable in B_4 with
four state variables x^0, x^1, x^*, $\bar{x}^*$ in B_2
(here superscripts cannot be misunderstood
as exponentials due to the idempotent law of
Boolean algebra, subscripts are reserved for
numbering variables in B_4) such that

$x=0$ in B_4 iff $x^0=1, x^1=x^*=\overline{x}^*=0$ in B_2

$x=1$ in B_4 iff $x^1=1, x^0=x^*=\overline{x}^*=0$ in B_2

$x=D$ in B_4 iff $x^*=1, x^0=x^1=\overline{x}^*=0$ in B_2

$x=\overline{D}$ in B_4 iff $\overline{x}^*=1, x^0=x^1=x^*=0$ in B_2

then we call x^0, x^1, x^*, $\overline{x}^*$ the _components_
of the _vector_ x in B_4. Obviously we have

$$x=x^0.0+x^1.I+x^*.D+\overline{x}^*.\overline{D}$$

which is called the _vector form_ of x in B_4

1.4 Star Algorithm

Given a Boolean function $F(x_1,\ldots,x_n)$, we
are going to derive the formulas for its com-
ponents F^0, F^1, F^*, F^*.

Theorem 1.2

For the basic Boolean functions $\overline{x}$, $x.y$,
$x+y$ we have

$(\overline{x})^0=x^1$, $(\overline{x})^1=x^0$, $(\overline{x})^*=\overline{x}^*$, $(\overline{\overline{x}})^*=x^*$

$(x.y)^0=x^0+y^0+x^*\overline{y}^*+\overline{x}^*y^*$ $(x.y)^1=x^1y^1$

$(x.y)^*=x^*y^1+x^1y^*+x^*y^*$ $(\overline{x.y})^*=\overline{x}^*y^1+x^1\overline{y}^*+\overline{x}^*\overline{y}^*$

$(x+y)^0=x^0y^0$ $(x+y)^1=x^1+y^1+x^*\overline{y}^*+\overline{x}^*y^*$

$(x+y)^*=x^*y^0+x^0y^*+x^*y^*$ $(\overline{x+y})^*=\overline{x}^*y^0+x^0\overline{y}^*+\overline{x}^*\overline{y}^*$

For any Boolean function F in B_4, if we
exchange all the compontents "*" and "-*"
of the input variables, then the formulas for
F^* and $\overline{F}^*$ are exchanged.

Since Boolean function is formed by ".",
"+", "-", using Theorem 1.2 iteratively we
can derive the formulas of F^0, F^1, F^*, $\overline{F}^*$
for any Boolean function F.

Let $N=\{1,2,\ldots,n\}$, (P,Q) be a partition
of N, (x_p, x_Q) be a partition of $\{x_1,\ldots,x_n\}$,
π be a Boolean vector in the subspace $B_2^{|P|}$.
Then for any Boolean function $F(x_1,\ldots,x_n)$
$=F(\mathbf{X})$, we have

Theorem 1.3   (Expansion)
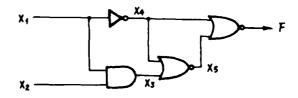
$$F^* = \sum_{\phi \neq P \subseteq N} \sum_{\pi \in B_1^{|P|}} (X_P^{\pi^*}) \overline{F(\overline{\pi},X_Q)} \cdot F(\pi, X_Q)$$

$$\overline{F}^* = \sum_{\phi \neq P \subseteq N} \sum_{\pi \in B_2^{|P|}} (X_P^{\pi^*}) F(\overline{\pi},X_0) \cdot \overline{F(\pi,X_Q)}$$

Where P is taken over all the subset of N,
π runs over all the Boolean vector space of
dimension $|P|$, $\overline{\pi}$ is the complement of π
taken coordinate by coordinate, $(x_p^{\pi^*})$ is
the product of those variables with supersc-
ripts "*" and "-*".

Example 1.1 For the Boolean function F de-
fined by the circuit shown in Fig 1.1, de-
rive the formula for F^*.

(1) Using Theorem 1.2

316

$$F^* = \overline{(x_4+x_5)}^* = (\overline{x}_4 \cdot \overline{x}_5)^*$$
$$= \overline{x}_4^* \overline{x}_5^1 + \overline{x}_4^1 \overline{x}_5^* + \overline{x}_4^* \overline{x}_5^*$$
$$= \overline{x}_4^*(x_3+x_4)^1 + \overline{x}_4^1(x_3+x_4)^* + \overline{x}_4^*(x_3+x_4)^*$$
$$= \overline{x}_4^*(x_3^1+x_4^1+x_3^*\overline{x}_4^*+\overline{x}_3^*x_4^*) + \overline{x}_4^1(x_3^*\overline{x}_4^1+\overline{x}_3^1x_4^*+x_3^*x_4^*)$$
$$+ \overline{x}_4^*(x_3^*\overline{x}_4^1+\overline{x}_3^1x_4^*+x_3^*x_4^*)$$

Form (1.1), we have $x_4^1\overline{x}_4^* = x_4^*\overline{x}_4^* = \overline{x}_4^1x_4^* = \overline{x}_4^1\overline{x}_4^* = 0$.

Thus $F^* = \overline{x}_4^* x_3^1 + x_3^* \overline{x}_4^* + \overline{x}_4^1 x_3^*$

$$= x_1^*(x_1 x_2)^1 + (x_1 x_2)^* x_1^* + x_1^1(x_1 x_2)^*$$
$$= x_1^* x_1^1 x_2^1 + (x_1^* x_1^1 + x_1^1 x_2^* + x_1^* x_2^*)x_1^*$$

$$+ x_1^1(x_1^* x_2^1 + x_1^1 x_2^* + x_1^* x_2^*)$$

For $x_1^* x_1^1 = 0, \; x_1^* x_1^* = x_1^*$, we have
$$F^* = x_1^* x_2^1 + x_1^1 x_2^* x_1^* x_2^*$$

(2) Using Expansion Theorem 1.3

$$F = \overline{x_1 + \overline{x_1 + x_1}x_2} = x_1 x_2$$

$$F(x_1,1) = x_1, \qquad F(1,x_2) = x_2$$

$$F(x_1,0) = F(0,x_2) = F(0,0) = F(0,1) = F(1,0) = 0$$

$$F(1,1) = 1.$$

$$F^* = x_1^* F(1,x_2)\overline{F(0,x_2)} + \overline{x}_1^* F(0,x_2)\overline{F(1,x_2)}$$
$$+ x_2^* F(x_1,1)\overline{F(x_1,0)} + \overline{x}_2^* F(x_1,0)\overline{F(x_1,1)}$$
$$+ x_1^* x_2^* F(1,1)\overline{F(0,0)} + x_1^*\overline{x}_2^* F(1,0)\overline{F(0,1)}$$
$$+ \overline{x}_1^* x_2^* F(0,1)\overline{F(1,0)} + \overline{x}_1^*\overline{x}_2^* F(0,0)\overline{F(1,1)}$$
$$= x_1^* x_2 + x_2^* x_1 + x_1^* x_2^*$$

the same as formula (1.3).

1.5 Front and Rear Components

Definition 1.6

If $x \in B_4$, then let $x^1 + x^* = \lfloor x$, $x^1 + x^* = x\rfloor$ and call $\lfloor x, \; x\rfloor$ the front and rear components of x respectively.

Property 1

$$\lfloor\overline{x} = x^0 + x^*, \qquad \overline{x}\rfloor = x^0 + x^*$$

Property 2

If $x \in B_2$, then $\lfloor x = x\rfloor = x$, $\lfloor\overline{x} = \overline{x}\rfloor = x$;

Property 3

Suppose $F(x_1,\dots,x_n)$ is a Boolean function, then

$$\lfloor F = F(\lfloor x_1,\dots,\lfloor x_n), \qquad F\rfloor = F(x_1\rfloor,\dots,x_n\rfloor);$$

Property 4

The multiplication can be shown by the symmetric Table 1.2

	x^0	x^1	x^*	$\overline{x}^*$	$\lfloor x$	$x\rfloor$	$\lfloor\overline{x}$	$\overline{x}\rfloor$
x^0	x^0							
x^1	0	x^1						
x^*	0	0	x^*					
$\overline{x}^*$	0	0	0	$\overline{x}^*$				
$\lfloor x$	0	x^1	x^*	0	$\lfloor x$			
$x\rfloor$	0	x^1	0	$\overline{x}^*$	x^1	$x\rfloor$		
$\lfloor\overline{x}$	x^0	0	0	$\overline{x}^*$	0	$\overline{x}\rfloor$	$\lfloor\overline{x}$	
$\overline{x}\rfloor$	x^0	0	x^*	0	x^*	0	x^0	$\overline{x}\rfloor$

Table 1.2

Property 5

$$(x \cdot y \cdot z)^* = x^* \lfloor \underline{y} | \underline{z} + y^* \lfloor \underline{x} | \underline{z} + z^* \lfloor \underline{x} | \underline{y},$$
$$(x + y + z)^* = x^* \overline{y}\rfloor \overline{z}\rfloor + y^*\overline{x}\rfloor \overline{z}\rfloor z^*\overline{x}\rfloor \overline{y}\rfloor.$$

Property 6

If equality

$$F(*, -*, \lfloor, \rfloor) = G(*, -*, \lfloor, \rfloor)$$

holds, then also holds

$$F(-*, *, \rfloor, \lfloor) = G(-*, *, \rfloor, \lfloor)$$

in which the components "*" and "-*", "⌊" and "⌋" are exchanged respectively. (e.g. see Property 1).

The advantage of using front and rear

components is that formulas (1.11) and (1.12) contain only n terms while the corresponding formulas using (1.3) and (1.4) should contain 2^n terms. Besides, in virtue of (1.9) the evaluation of these components of a Boolean function F in B_4 is simply the evaluation of F in B_2 with respect to the corresponding component of the input variables.

The introducing of the front and rear components needs four additional states for each variable, and thus needs more space to store them. But these four additional states are essentially the combinations of the components $0,1,*,-*$. If these eight states are fine coded, for example, if we use four bits to record the components of a variable x, i.e.

$$x^0 = x(1\ 0\ 0\ 0)$$
$$x^1 = x(0\ 1\ 0\ 0)$$
$$x^* = x(0\ 0\ 1\ 0)$$
$$\overline{x}^* = x(0\ 0\ 0\ 1)$$

then $\lfloor x = x^1 + x^* = x(0\ 1\ 0\ 0) + x(0\ 0\ 1\ 0) = x(0110)$, and $x\rfloor = x(0\ 1\ 0\ 1)$, $\lfloor \overline{x} = (1\ 0\ 0\ 1)$, $\overline{x}\rfloor = (1010)$. The operations can be executed parallely without increasing the time complexity, e.g.

$$\lfloor x \cdot x\rfloor = x(0110) \cdot x(0101) = x(0100) = x^1$$
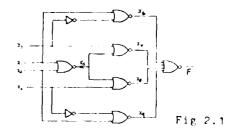
The application of Four-Valued Logic will depend upon the meaning which we shall assign to the front and rear components of a variable. Here three kinds of applications are shown in the following sections.

In a logic circuit. Let us associate to each wire w a variable x=(a,b) in B_4, where a (b) is the value of w in B_2 when the circuit is normal (faulty). $x^*=1$ ($\overline{x}^*=1$) or x=D ($\overline{D}$) means the wire w is affected by a fault in the circuit under the given input values, otherwise x is not affected and then takes the value in B_2. For the output function F, $F^*=1$ or $\overline{F}^*=1$ means that the difference of the normal and faulty effects has been sensitized to the output, hence the corresponding fault can be detected. Here we illustrate it by examples and ignore the theorems.

Example 1.2

In the circuit shown in Fig. 2.1, find out the complete test set for the stuck-at-0 fault at x_5

Fig 2.1

The fault propagtes through x_7, x_8 to F, so x_5, x_7, x_8, $F \in B_4$ and $x_1, x_2, x_3, x_4, x_6, x_9$, $\in B_2$. Using Star-Algorithm

$$F^* = (\overline{x_6 + x_7 + x_8 + x_9})^* = (\overline{x}_6 \overline{x}_7 \overline{x}_8 \overline{x}_9)^*$$
$$= \overline{x}_7^* \overline{x}_6 \lfloor \overline{x}_8 \overline{x}_9 + \overline{x}_8^* \overline{x}_8 \rfloor \overline{x}_7 \overline{x}_9$$
for $\overline{x}_6^* = \overline{x}_9^* = 0$ and $\lfloor \overline{x}_6 = \overline{x}_6$, $\lfloor \overline{x}_9 = \overline{x}_9$

from property 2 in section 1.3

$$F^* = (x_1+x_5)^*(\bar{x}_1+x_2)\lfloor(x_4+x_5)(x_3+\bar{x}_4)$$
$$+ (x_4+x_5)^*(\bar{x}_1+x_2)\lfloor(x_1+x_5)(x_3\,\bar{x}_4)$$

From (1.4), $(x_1+x_5)^* = x_5^*\bar{x}_1$.

From (1.9) $\lfloor(x_1+x_5) = \lfloor x_4 + \lfloor x_5 = x_4 + \lfloor x_5$.

$$F^* = x_5^*\bar{x}_1(x_4+\lfloor x_5)(x_3\cdot\bar{x}_4 + x_5^*\bar{x}_4(\bar{x}_1+x_2)(x_1+\lfloor x_5);$$

From (1.10), $x_5^* \cdot \lfloor x_5 = x_5^*$,

$$x_5^*(x_4 + \lfloor x_5) = x_5^* x_4 + x_5^* = x_5^*$$

$$F^* = x_5^*\bar{x}_1(x_3+\bar{x}_4) + x_5^*\bar{x}_4(\bar{x}_1+x_2)$$

$$= x_5^*(\bar{x}_1 x_3 + x_2\bar{x}_4 + \bar{x}_1\bar{x}_4) .$$

This means the faulty difference at x_5 can be sensitized to the output F in the same sence (i.e.D) under the input condition:

$$\bar{x}_1 x_3 + x_2\bar{x}_4 + \bar{x}_1\bar{x}_4 = 1 .$$

Sice the fault is x_5: s-a-0, the test must make $x_5 = 1$. Thus the complete test set for x_5 s-a-0 is

$$T = (\overline{x_2+x_5})(\bar{x}_1 x_3 + x_2\bar{x}_4 + \bar{x}_1\bar{x}_4) = \bar{x}_1\bar{x}_2\bar{x}_5\bar{x}_4$$
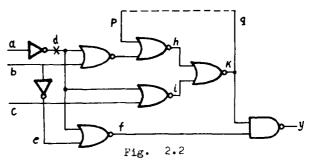
From (1.11) and (1.12) the output of a gate has value D iff at least one input x_i has value D while the other inputs x_j may or may not have value D. In this case we call x_i the __main__ sensitizing branch. From (1.11) and (1.12), we can take each x_i in turn as main sensitizing branch.

Similarly Star Algorithm can be applied to the test generation for the multipe fault in combinational circuit, hence can also be applied to that of sequential circuit.

Example 1.3

Fig. 2.2 is an asynchronous sequential circuit. If the feedback line, shown by the dotted line, be cut off, then Fig. 2.3 becomes the model of the same sequential circuit in one time frame, say T; p_T and q_T become pseudo-input and pseudo-output of the T-th time frame respectively.

Fig. 2.2

Suppose the fault d:s-a-1 occurs, then there are three paths for propogating value D from d to y, namely

L_1: d-g-h-k-y L_2: d-i-k-y; L_3 d-f-y.

these variables $d,g,h,i,f,y \in B_4$, while $a,b,c,p \in B_2$

(1) Taking L_1 as the main sensitizing path, there are four inversions along L_1, so a $\bar{D}$ at d can cause a $\bar{D}$ at y. We expand $\bar{y}_T^{-*}$.

$$\bar{y}_T^{-*} = (k_T f_T)^* = k_T^* \lfloor f_T = (\bar{h}_T\bar{i}_T)^* \lfloor \bar{d}_T b_T$$
$$= \bar{h}_T^* \lfloor \bar{i}_T (\lfloor \bar{d}_T \cdot b_T) = g_T^* \lfloor \bar{p}_T \lfloor \bar{i}_T \lfloor \bar{d}_T b_T$$
$$= \bar{d}_T^{-*}\bar{b}_T \lfloor \bar{p}_T \lfloor i_T \lfloor d_T b_T = 0 .$$

This means we can not propogate $\bar{D}$ through this path.

(2) There are three inversions along L_2, so we need expand y_T^*

$$y_T^* = \bar{k}_T^* \lfloor f_T \rfloor = (i_T^* h_T \lfloor) (\bar{d}_T \lfloor b_T) = (\bar{d}_T^{-*}\bar{c}_T) h_T \lfloor \bar{d}_T \lfloor b_T = 0 .$$

because at the fan-out point d, we have $d_T^* d_T \lfloor = 0$.

(3) There are two inversions along L_3, we expand $\bar{y}_T^{-*}$

$$\bar{y}_T^* = f_T^* \lfloor k_T = (\bar{d}_T^* b_T) \lfloor \bar{h}_T \lfloor \bar{i}_T = \bar{d}_T^* b_T (\lfloor p_T + \lfloor g_T)(\lfloor c_T + \lfloor d_T)$$

$$= \bar{d}_T^* b_T (p_T + \bar{b}_T \bar{d}_T)(c_T + \lfloor d_T = \bar{d}_T^* b_T c_T \lfloor p_T$$

That means value $\bar{D}$ can be driven to y. Since d:s-a-1, the test should make $d_T=0$, thus

$$Test = \bar{d}_T b_T c_T \lfloor p_T = a_T b_T c_T \lfloor p_T$$

the Pseudo-input p_T is the pseudo-output q_{T-1} in the T-1 th frame

$$\lfloor q_{T-1} = \lfloor \bar{h}_{T-1} \lfloor \bar{i}_{T-1} = (\lfloor g_{T-1} + \lfloor p_{T-1})(\lfloor d_{T-1} + c_{T-1})$$

$$= (\bar{b}_{T-1} \lfloor \bar{d}_{T-1} + \lfloor p_{T-1})(\lfloor d_{T-1} + c_{T-1})$$

$$= \bar{b}_{T-1} c_{T-1} \lfloor \bar{d}_{T-1} + \lfloor p_{T-1}(\lfloor d_{T-1} + c_{T-1})$$

$$= \bar{b}_{T-1} c_{T-1} a_{T-1} + \lfloor p_{T-1}(\lfloor d_{T-1} + c_{T-1})$$

If we take the first term $a_{T-1} b_{T-1} c_{T-1}$, which is independent of the pseudo-input p, then we have one

$$Test = a_T b_T c_T \lfloor p_T = a_T b_T c_T a_{T-1} \bar{b}_{T-1} c_{T-1}$$

Let T-1=1 be the first time frame, then one test sequence obtained is

$$Test = a_1 \bar{b}_1 c_1 a_2 b_2 c_2$$

which detects the single fault d:s-a-1 no matter what initial state the circuit is. The other solutions are included in the second term of $\lfloor q_{T-1}$, which depends upon p_{T-1} hence upon the inputs in the time frames before T-1.

III HAZARDOUS TEST GENERATION [3]
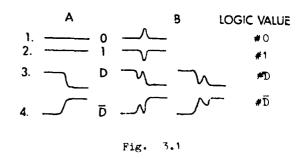
Let us interpret the front and rear values of a variable in B_4 respectively as the voltage levels of a wire before and after a certain moment, then D and $\bar{D}$ become the fall (step-down) and rise (step-up) of a rectangular pulse. Therefore four-valued logic can be applied to the dynamic study of the logic circuit.

Here our attention is focused on the hazards and the dynamic test generation.

The waveforms of the static 0-hazard (denoted by $\neq 0$), the static 1-hazard ($\neq 1$), the dynamic step-down hazard ($\neq D$), the dynamic step-up hazard ($\neq \bar{D}$) are shown in Fig. 3.1. If we only use four-valued logic, they belong to the sets 0, 1, D, $\bar{D}$ respectively as shown in the figure. In order to identify a hazard $\neq$s, s$\in$S and S={ 0,1,*,-* }, we need to expand the corresponding component F^s, (s$\in$S), of the output function F of the combinational circuit by Star Algorithm and ignore the other three. Then we check each term of F^s, whether it causes the output to issue the corresponding hazard or not. To verify this we should deal with six-valued logic and eight-valued logic, see [5]. After that we can divide all the terms of F^s into two parts, the hazard part F^{sh} and the hazard-free part $F^{s\bar{h}}$

$$F^s = F^{sh} + F^{s\bar{h}} \qquad s \in \{0, 1, *, -*\}$$

	A		B		LOGIC VALUE
1.	———	0	⋀		$\neq 0$
2.	———	1	⋁		$\neq 1$
3.	⎤_	D	⋀⌐	⌐⋀	$\neq D$
4.	_⎡	$\bar{D}$	⌐⋁	⋁⌐	$\neq \bar{D}$

Fig. 3.1

When a logic fault g occurs, the output function becomes G, we can find similarly

$$G^s = G^{sh} + G^{s\bar{h}} \qquad s \in \{0, 1, *, -*\}.$$
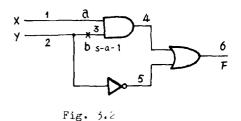
If the normal circuit is hazard-free while the faulty circuit is hazardous, or if the normal is hazardous while the faulty is hazard-free, then we can distinquish the faulty from the normal. So the complete hazardous test set of the fault g is

$$T_g^H = \sum_{s \in S} (F^{sh}G^{s\bar{h}} + F^{s\bar{h}}G^{sh}).$$

Example 3.1

For the circuit shown in Fig. 3.2, the normal function $F = xy + y$, we have

Fig. 3.2

$F^{0h} = 0$, $\qquad\qquad F^{0\bar{h}} = \bar{x}y$

$F^{1h} = y^* x + x^* y^* + \bar{y}^* x + \bar{x}^* \bar{y}^* \qquad F^{1\bar{h}} = x + \bar{y} + x^* \bar{y} + \bar{x}^* y$

$F^{*h} = x^* \bar{y}^* \qquad\qquad F^{*\bar{h}} = x^* y + \bar{y}^* \bar{x}$

$\bar{F}^{*h} = \bar{x}^* y^* \qquad\qquad \bar{F}^{*\bar{h}} = \bar{x}^* y + y^* \bar{x}$

For the fault b:s-a-1 the faulty output function $G = x + \bar{y}$,

$G^{0h} = 0 \qquad\qquad G^{0\bar{h}} = \bar{x}y$

$G^{1h} = x^* y^* + \bar{x}^* \bar{y}^* \qquad G^{1\bar{h}} = x + \bar{y} + x^* \bar{y} + y^* x + \bar{x}^* \bar{y} + \bar{y}^* x$

$G^{*h} = 0 \qquad\qquad G^{*\bar{h}} = x^* y + \bar{y}^* x + x^* y^*$

$G^{*h} = 0 \qquad\qquad G^{*\bar{h}} = \bar{x}^* y + y^* \bar{x} + x^* y^*$

The complete hazardous test set is

$$T_g^H = \sum_{s \in S} (F^{sh}G^{s\bar{h}} + F^{s\bar{h}}G^{sh})$$

$$= y^* x + \bar{y}^* x + x^* \bar{y}^* + \bar{x}^* y^*$$

The first term $y^* x$ comes from the term $F^1 h_G^{1\bar{h}}$, that means the input x=1, y=D is a test which causes the notmal circuit to produce a static 1-hazard while the faulty circuit is 1-hazard free. We call this test a 1-hazard test.

Example 3.2

Consider a circuit with high redundancy shown in Fig. 3.3. Many single stuck faults in it are statically undetectable, but all of them can be detected by 1-hazard tests. this fact is shown in Table 3.1

Fig. 3.3
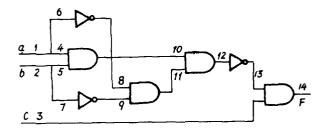
Leads	Single stuck fault detected													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Static Tests		1								1	1	1	0	1
		0												0
1-hazard test a b c	1	1			1	1							1	1
	0	0		0	0			0	0	0	0	0		
ā b c					0		1							
a b̄ c					0		1							
ā b̄ c			1											
a b̄ c̄			1											

Table 3.1

By using the hazardous test we obtained some results:

(1) If the existence of a given redundant wire R in the circuit C is to eliminate the logic hazard, then the statically undetectable fault at R is hazardously detectable.

(2) In a non-redundant combinational circuit,

every intragate bridging fault is either statically or hazardously detectable.

IV TRANSITION LOGIC [4]

In the classical two-valued logic, the truth value of each proposition x is taken from $B_2 = \{0, 1\}$. Usually these two values are used to represent two static states of an object in the real world. Now if the value of x is taken from B_4, i.e. beside the stable states "0" and "1", we have two additional states: $D=(1,0)$ the transitional state from "1" to "0", and $\bar{D}=(0,1)$ the transitional state from "0" to "1", then this four-valued logic can be used to describe (though roughly) the transitional phenomena or the dynamic behavior of the object in the real world. For example, if x represents water, $x=0$ represents x in the liquid state and $x=1$ in the vapor state; these are stable states. Then $x=D=(1,0)$ means x is in the condensation state and $x=\bar{D}=(0, 1)$ means x is in the evaporation state; these are transitional states. The generalization of the classical logic to the transition logic is essential, it seems to be a leap in the process of cognition from the static field into the dynamic field.

4.1 Propositional Calculus and Predicate Calculus

In the definition 1.2 we add two more operations "→" and "⇆" such that

$$F \rightarrow G = F + G, \qquad F \rightleftarrows G = FG + \bar{F}\bar{G}$$

where F and G are expressions, then the generalized expressions are the formulas for general propositions. Definition 1.3 and Theorem 1.1 for equivalence still holds, espesially if $F(x_1,\ldots,x_n) = G(x_1,\ldots,x_n)$ for all $x_i \in B_2$, then $F(x_1,\ldots,x_n) \sim G(x_1,\ldots,x_n)$ and therefore $F(x_1,\ldots,x_n) = G(x_1,\ldots,x_n)$ for all $x_i \in B_4$. Namely the proposition calculus in the classical logic can be generalized to the transition logic without any change.

If we know the equality
$$F(x_1,\ldots,x_n) = G(x_1,\ldots,x_n)$$
holds for all $x_i \in B_2$, then we can extend it to B_4. In fact, we can add two additional states $D=(1, 0)$, $\bar{D}=(0, 1)$ to each variable and let $1=(1,1)$, $0=(0,0)$, i.e., let all $x_i \in B_4$. Then for each logic operation O in F and G, we extend its meaning by
$$(a,b)O(c,d) = (aOb, cOd).$$
Since $B_4 = B_2 * B_2$, we assert that this equality also holds after the extension to B_4

Example 4.1

Let the propositions P, Q and R be defined in B_2 as the following

		0	1
P:	state of the substance is	liquid	vapor
Q:	container temperature is	low	high
R:	pressure in the container is	low	high

It is easy to verify that $P=Q\bar{R}$ holds in B_2. Now let us extend the definitions to B_4

	0	1	D	$\bar{D}$
P: state of the substance is	liquid	vapor	evaporation	condensation
Q: temperat. of the container is	low	high	increasing	decreasing
R: pressure in the container is	low	high	increasing	decreasing

The extended operation $P=Q\bar{R}$ in B_4 is shown in Table 4.1

$P=Q\bar{R}$	R:	0	1	D	$\bar{D}$
Q:0		0	0	0	0
1		1	0	D	D
D		D	0	0	D
$\bar{D}$		D	0	D	0

Table 4.1

It is also reasonable, for example, if we use Star Algorithm

$$P^* = (Q\bar{R})^* = Q^*\bar{R} + Q\bar{R}^* + Q^*\bar{R}^*$$

That means there are only three possibilities to cause this substance evaporated

$Q=D$ and $R=0$; i.e. keeping low pressure and inseasing the temperature.

$Q=1$ and $R=\bar{D}$; i.e. keeping high temperature and decreasing the pressure.

$Q=D$ and $R=\bar{D}$; i.e. incressing the temperature and decreasing the pressure.

Since B_4 has more states than B_2, what the former reflects seems richer and more frofound than the latter.

Similarly let

$$(\forall x)F(x) = \prod_{x \in D} F(x)$$
$$(\exists x)F(x) = \sum_{x \in D} F(x)$$

Where D is the domain of x, we can generalize the predicate calculus to B_4.

Theorem 4.1 Let S be a set of clauses, then S is always true (or non-satisfiable) on B_4 iff S is always true (or satisfiable) on B_2

4.2 Logical Inference

From Theorem 1.1, the rules for logical inference on B_2 can be generalized to B_4

Lemma 1 The formula G is a logical consequence of formula F in B_4 iff G is a logical consequence of F in B_2

Lemma 2 Given a set S of clauses, then a clause C is a logical prime consequence of S in B_4 iff C is a logical prime consequence of S in B_2

Let S be a set of clauses, R(S) be the resolution of S, i.e. the set consisting of the resolution formulas of all the clauses of S and the clause pairs of S. The n-th resolution $R^n(S)$ of S is defined by

$$R^0(S)=S \text{ and } R^n(S)=R(R^{n-1}(S)).$$

Theorem 4.2 (The Completeness of Resolution Principle in B_4)

If a clause C is a logical prime consequence of clause set S, then there is a $n \geqslant 0$ such that $C \in R^n(S)$.

In B_2 if G is a logical consequence of F, then $G \geqslant F$. The following shows that this relation also holds in B_4, though B_4 is semi-

323

ordered.

Definition 4.1

Elements a $b \in B_4$ have the relation $b \geqslant a$ iff $a.b=a$.

Obviously $a \geqslant a$; if $b \geqslant a$, $c \geqslant b$, then $c \geqslant a$.

Lemma 4.3

If formula $F \Rightarrow G$ in B_4, then $F \geqslant G$ in B_4

Lemma 4.4

Let C_1 and C_2 be two clauses in B_4, $R(C_1, C_2)$ is any resolution formula of C_1 and C_2, then

$$R(C_1, C_2) \geqslant C_1 \cdot C_2$$

Theorem 4.3

Let S be a set of clause $C_1, C_2, \ldots, C_m$, C^n be any clause in the set $R^n(S)$. Then for every $n \geqslant 0$, we have

$$C^n \geqslant \prod_{i=1}^{m} C_i .$$

V   CONCLUSION

In this paper, we just figure out our four-valued logic and three kinds of its applications. This four-valued logic B_4 differs from the classical four-valued logic L_4 by (1) B_4 is semi-ordered, (for D and $\bar{D}$ are not comparable), thus B_4 can not be used to form quadnary number system instead of L_4 which is well ordered.
(2) L_4 is but a distributive lattice while B_4 is a Boolean algebra. So that many advantages of B_4 are inherented from the tranditional Boolean algebra which is familiar to everyone.

In test generation, the Star expansion provides a cause-effect relation which tells us how the external causes (x_i^*'s, the faulty signals) propogate through the network and produce an effect to the output. It seems better to add some state variables to describe the internal cause (faults in the circuit) and obtain a more complete cause-effect relation.

For the second interpretation in III, another potential application of B_4 is the analysis and synthesis of the sequential circuit. Here to solve the Boolean equation in B_4 is a difficult problem, because there are proper zero divisors D and $\bar{D}$ in B_4 (for $D * \bar{D} = 0$).

For the transition logic, B_4 is but a qualitative description. our goal is to find certain kind of quantitative description which is more precise than B_4.

However, our four-valued logic is at the infant stage, what has been done seems too less than what we have to do in the future.

REFERENCES

1. T.H. Chen, "Four-Valued Logic and Star Algorithm", Chinese Computer Journal, Vol. 2, No.4, pp 243-264, Oct. 1979.

2. Z.M. Zhang and T.H. Chen, "A Near Optimal Algorithm for Test Generation in Sequential Circuit-Star Algorithm", Symposium on Fault Tolerant Computing 1981, held in Portland, Main, USA, June 1981, pp233-237.

3. Y.G. Yuan and T.H.Chen, "The dynamic Testing of Combinational Logic Network", Symposium on Fault Tolerant Computing

1982, held in Santa Monica, California, USA.
Jun. 22-24 1982, pp173-180.

4. T.H. Chen, "Transition Logic", Journal
   of Chongqing University, No. 3 1981,
   pp194-197. (in Chinese)

5. Z.M. Zhang and T.H. Chen, "Application of
   Star Algorithm to Multiple Fault Detec-
   tion in Combinational Circuit", Journal
   of Chongqing University, No.3, 1981,pp41-
   61. (in Chinese)

6. Y.G. Yuan and T.H. Chen," Hazard Identi-
   ficatin in Combinational Logic Circuit",
   Journal of Chongqing University, No. 3,
   1981, pp176-195. (in Chinese)

7. Z.M. Liu; Y.G. Yuan and T.H. Chen,"vec-
   tor Expansion Transformation of Logic
   Algebra". ISMVL-83

Session 7B
Algebra II

Mx, A MIX-VALUED ALGEBRA

Michael Sinutko Jr.

Department of Defense
Post Office Box 1747
Washington, D.C. 20013
Phone: (202)694-3361

James H. Pugsley

Electrical Engineering Dept.
University of Maryland
College Park, Maryland 20742
Phone: (301)454-6862

Abstract

A combinatorial "mix-valued" algebra,
denoted by "Mx," operates on variables
representing multivalued signals and buses
of any width (including width = 1). A bus
of signals is represented as a single mul-
tivalued variable. All variables in Mx can
range over values and sets of values and
are not required to range over the same
. The function set proposed for Mx
includes relational, set theoretic, and
existential operators. The usual two-
valued Boolean algebra is a subalgebra of
Mx when all variables are binary and from
the same set of values.

Only combinatorial memoryless topics
are discussed, but sequential and memory
circuits composed of Mx operations are
known.

Mx is useful for compact technology-
independent representation of digital sys-
tems during the design process.

1.0 Introduction

Mx is a technology-independent mix-
valued algebra[9] for describing and design-
ing digital circuits* interconnected by
combinations of multivalued signals and
buses of any mix of widths including
width = 1. A p-wide input or output bus of
q-valued signals is treated as a single
multivalued variable ranging over up to q^p
elements. Variables in Mx can range over
dissimilar, heterogeneous sets.

All functions in Mx must conform to
the structure given in Section 2.1, but are
otherwise arbitrary. Ten such Mx-functions
are proposed and described. Other conform-
ing functions can be used, but properties
such as functional completeness are the
responsibility of their designer.

---

* We pronounce Mx as "mix." "Circuit" or
"network" means combinatorial circuit or
network.

For experienced logic designers, the
AND, OR, and NOT Mx-operators are 2-valued
Boolean operators when applied to binary
variables over the same set. Also, the
proposed symbolism and design procedure
resemble those of 2-valued Boolean algebra.

Each Mx-gate g (function realizor)

a) accepts inputs and generates outputs
from its own independent "reference
set," denoted by r_g (a reference set may
be ordered or not, as needed)

b) accepts and generates a "null variable"
denoted by $\emptyset$.

A reference set specifies all I/O (input
and/or output) elements recognized by a
Mx-gate ($\{0,1\}$ is a 2-valued Boolean gate
"reference set"). Mx-gates having differ-
ing reference sets can interact. A null
variable ($\emptyset$) has zero cardinality and
ranges over the heterogeneous set of value-
less members[9]. (The member instance of the
null variable $\emptyset$ is a member of every set.)
"Null" hereafter means null variable unless
stated otherwise.

A $\emptyset$-bearing input is interpreted in
one of two ways at Mx-gate inputs depending
on the gate type: a) it has vanished, need-
ing no consideration in the gate output
determination; or b) it appears to have no
value. The "existential" Mx-operators
implement (b) by waiting for a non-$\emptyset$ on
all, none, or exactly one of the inputs
before generating an output.**

Mx never requires more equations than
2-valued Boolean algebra to describe a
binary circuit with busing because Mx-
equations are written for a whole bus at a
time; not for each bus signal component.
Using Mx generally results in a very com-

** The "waiting" behavior is similar to
that found prior to the "firing" of Petri
net "transitions."[1] "Transitions" are unde-
fined for input values not within the
operator value-universe. Mx operations are
defined under such a condition.

pact set of equations which fully describe the intended behavior of the network. The compactness becomes dramatic with the increased use of busing.

A Mx-logic design needs no redesign to change realizations; only a new translation of the logic design into hardware. Standard implementation circuits for the Mx-gates in various technologies may evolve to fill this need.

The reader is invited to preview the application Sections 2.3 and 2.4 for samples of Mx's operability.

2.0 Formal Definition of Mx

Mx is defined as the pair (S,F) wherein

S is a totally ordered set, the union of the sets I, O, and R_g, and further decomposed such that no member of S is a set, where

$I = \{X_1, X_2, \ldots\}$ is the set of all input sets X_m where $X_m = \{x_1, x_2, \ldots, x_n\}$, a set of input variables, for arbitrary m and n,

$O = \{z_1, z_2, \ldots\}$ is the set of all output variables z_m,

$R_g = \{R_{g1}, R_{g2}, \ldots\}$ is the set of all sets of reference sets r_{gi}, and $R_{gm} = \{r_{g1}, r_{g2}, \ldots, r_{gn}\}$ for arbitrary i, m, and n, and

F is a set of functions from I to O.
——End of Definition——

Reference sets, set "I" and set "O" are formally independent of each other, but to date, meaningful applications of F have only involved I and O sets which intersect R_g.

A $\emptyset$ can render its sending and receiving Mx-operators temporarily non-associative and non-distributive; it implies an into map. Thus F is conditionally associative and distributive.

2.1 The Mx "General Function Structure" (GFS)

The following definition is in the Mx GFS, to which all functions in F must conform. "fi" and "ci" are function and condi-

tions "i," respectively. Note that although $x_i \in X$, $x_i \subseteq r_g$ is used because generally, x_i can be a set which is a member of X, and a subset but not a member of r_g. x_i can be both a member of X and r_g.

$$\#(X) \equiv \left\{ \begin{array}{l} \left\{ \begin{array}{l} \forall x_i \in X \exists : x_i \subseteq r_g, \text{ only one of:} \\ f1 \text{ iff } c1 \text{ are met,} \\ f2 \text{ iff } c2 \text{ are met,} \\ \quad \cdot \\ \quad \cdot \\ \quad \cdot \\ fn \text{ iff } cn \text{ are met.} \end{array} \right\} \\ \text{the null variable } (\emptyset) \text{ otherwise,} \end{array} \right.$$

where "#" is an arbitrary gate operation, X is the input set, $X = \{x_1, x_2, \ldots, x_m\}$, and $fj \subseteq r_g$ for $j \in \{1, 2, \ldots, n\}$. "$\forall x_i \in X \exists : x_i \subseteq r_g$" is an input filter operation.
——End of Definition——

Let $x_i \in X$ be variables in an input set X, $i \in \{1, 2, 3, \ldots\}$. Define as an "alien" any input instance not a subset of r_g (i.e., an $(x_i \in X) \not\subseteq r_g$). If one or more aliens are present, the GFS's input filter causes the gate to yield a null. In the presence of a non-alien-bearing input, the GFS requires the output to be one of a set of user-devised conditional functions or a null.

2.2 The Proposed Function Set

Reasons for using a given 2-valued Boolean operation vary. For example, "x AND y" can mean greatest lower bound, product, min, intersection, existential (or, sentential[2]) truth or simultaneity, and maybe others. Similar statements can be made for OR and NOT. The following table lists some such interpretations on a 2-valued Boolean AND operation over {0,1}. Similar tables can be produced for OR and NOT.

x	y	AND (x,y)	product (x,y)	min (x,y)	intersect (x,y)	existential truth	simultaneity
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1

These ambiguities in 2-valued Boolean operator use suggest three function <u>classes</u> — AND, OR, and NOT — which are basic to the logic design procedure discussed in Section 2.3.

The proposed set of Mx-functions is F ≡ {AND(X), OR(X), NOT(X), UNION(X), INTERSECT(X), COMPLEMENT(X), ANDe(X), ORe(X), NOTe(X), EXIST(X)}. F addresses relational, set-theoretic, and existential domains. (The latter is concerned with the presence, or existence, of elements instead of their manipulation.) F members are further assigned to the three function classes as shown in the following table.

Mx-gate	generic identity	class
AND	MIN, algebraic AND	AND
OR	MAX, algebraic OR	OR
NOT	INVERT, algebraic COMPLEMENT	NOT
INTERSECT	set-theoretic AND	AND
UNION	set-theoretic OR	OR
COMPLEMENT	set-theoretic NOT	NOT
ANDe	existential AND	AND
ORe	existential OR	OR
NOTe	existential NOT	NOT
EXIST	existential acknowledgement	AND

To understand some of the proposed Mx-gates first requires the following definitions for "full set" (converse of the empty set), and the "atomizer function" (which decomposes a heterogeneous set so that no member of the resultant set is a set).

Full Set - A set denoted by $\underline{U}$ and composed of

a) all members of a given countable set

b) all subsets of that countable set excluding all its single member subsets.

Example 2.2-1
Assume the universe is the set {a, b,c,d}. Then (a) of the full set definition contributes the elements a, b, c, and d to the full set, (b) contributes {a,b},

{a,c}, {a,d}, {b,c}, {b,d}, {c,d}, {a,b,c}, {a,b,d}, {b,c,d}, {a,c,d}, and {a,b,c,d}. The full set is then $\underline{U}$ = {a,b,c,d,{a,b}, {a,c},{a,d},{b,c},{b,d},{c,d},{a,b,c}, {a,b,d},{b,c,d},{a,c,d},{a,b,c,d}}.
————End of Example 2.2-1————

Atomizer Function   If S = {s_1, s_2, ...,s_m}, A(S) denotes the "atomizer" function on S. If $\underline{U}$ is S's full set, "~" is set difference, and e is an element (= irreducible set member = atom), let $\underline{u} \subseteq \underline{U}$ ∋: {$\underline{U}$ ~ $\underline{u}$} contains no elements and e∈$\underline{u}$. Then, A(S) ≡ A:S ⟶ $\underline{u}$ is one-one.

Example 2.2-2
Let S = {1,3,{6,7,8},{},2,{4,{5, 9}},0} where in this example, ∅ denotes only the null value[9]. Then A(S) = {1, 3,6,7,8,∅,2,4,5,9,0} = {1,3,6,7,8,2,4, 5,9,0}. If S⊆$\underline{u}$, A(S) = S, where $\underline{u}$ is from the atomizer function definition.
————End of Example 2.2-2————

If X is a set containing the null set as a member, note that A(X) defines the null value given the null set operand.

The ten definitions for Mx operations, beginning at the bottom of this page, conform to the GFS. In those Definitions, X = {$x_1, x_2, ..., x_n$} is the input set to a Mx-gate and x_i∈X.

The NOT(X) definition needs a totally ordered r_g whose least member behaves like a zero. Its next higher member must behave like a one, the next higher must behave like a two, etc.[9]

"~" is set subtraction in the COMPLEMENT definition.

$$\textbf{UNION}(X) \equiv \begin{cases} union(\{A(x_1)\},\{A(x_2)\},...,\{A(x_n)\}) & \text{if } \forall x_i \in X, \ x_i \subseteq r_g \\ \emptyset & \text{otherwise} \end{cases}$$

$$\textbf{AND}(X) \equiv \begin{cases} min(A(X)) & \text{if } \forall x_i \in X, \ x_i \subseteq r_g \\ \emptyset & \text{otherwise} \end{cases}$$

$$\textbf{INTERSECT}(X) \equiv \begin{cases} intersection(\{A(x_1)\},\{A(x_2)\},...,\{A(x_n)\}) & \text{if } \forall x_i \in X, \ x_i \subseteq r_g, \\ \emptyset & \text{otherwise} \end{cases}$$

$$\textbf{COMPLEMENT}(X) \equiv \begin{cases} \{r_g \sim union(\{A(x_1)\},\{A(x_2)\},...,\{A(x_n)\})\} & \text{if } \forall x_i \in X, \ x_i \subseteq r_g \\ \emptyset & \text{otherwise} \end{cases}$$

$$\textbf{NOT}(X) \equiv \begin{cases} max(r_g) - A(X) & \text{if } \forall x_i \in X, \ x_i \subseteq r_g \\ \emptyset & \text{otherwise} \end{cases}$$

$$\textbf{OR}(X) \equiv \begin{cases} max(A(X)) & \text{if } \forall x_i \in X, \ x_i \subseteq r_g \\ \emptyset & \text{otherwise} \end{cases}$$

330

$$\text{ORe}(X) \equiv \begin{cases} x_1 \text{ if } x_1 \neq \emptyset,\ x_2 = x_3 = x_4 = \ldots = \emptyset \text{ and } x_1 \subseteq r_g, \\[4pt] x_2 \text{ if } x_2 \neq \emptyset,\ x_1 = x_3 = x_4 = \ldots = \emptyset \text{ and } x_2 \subseteq r_g, \\[4pt] \quad\vdots \\[4pt] x_n \text{ if } x_n \neq \emptyset,\ x_{n-1} = x_{n-2} = \ldots \\[4pt] \qquad = x_{n+1} = x_{n+2} = \ldots = \emptyset,\ \text{and } x_n \subseteq r_g, \\[4pt] \emptyset \text{ otherwise} \end{cases}$$

$$\text{ANDe}(X) \equiv \begin{cases} X \text{ if } \forall x_i \in X,\ x_i \subseteq r_g,\ \text{and } x_i \neq \emptyset \\[4pt] \emptyset \text{ otherwise} \end{cases}$$

$$\text{NOTe}(X) \equiv \begin{cases} r_g \text{ if } x_1 = x_2 = \ldots\ x_n = \emptyset \\[4pt] \emptyset \text{ otherwise} \end{cases}$$

$$\text{EXIST}(X) \equiv \begin{cases} \max(r_g) \text{ if } \forall x_i \in X,\ x_i \subseteq r_g \text{ and } x_i \neq \emptyset \\[4pt] \emptyset \text{ otherwise} \end{cases}$$

EXIST(X) is also written as "Ee(X)" to identify its class.

2.2.1 Properties of the Proposed Function Set

Mx uses $\{\cdot, +, ', \hat{}, \wedge, \vee, ", \&, \bullet, \not\bullet, \exists\}$ as algebraic "connectives" corresponding to {AND,OR,NOT,NOT,INTERSECT,UNION, COMPLEMENT,ANDe,ORe,NOTe,Ee}, respectively. (Note the two "NOT"s in the latter set. The apostrophe-type connective is used only when all variables are 2-valued and range over the same set.)

The execution precedence for Mx-expressions is:

1) The usual execution ordering applies for parenthetic statements in Mx-expressions.

2) Within each parenthetic statement, the operation precedence is, ordered left-to-right: $', \hat{}, ", \not\bullet, \exists, \cdot, +, \wedge, \vee, \&, \bullet$. (By this ordering, $x'y \neq x\hat{}y$ and $x'y = (x\hat{})y$.)

Following are some F properties which may be useful in manipulating Mx expressions. More are in the original work on Mx[9].

1) $\forall(x,y) \subseteq r_g$ for a given Mx-gate of function $*$ where $* \in \{\cdot, +, \hat{}, \wedge, \vee, ", \&, \bullet, \not\bullet, \exists\}$, $x*y = y*x$ ($x'y$ as a non-unary operation is undefined).

2) $\forall(x) \subseteq r_g$ for a given Mx-gate of function $*$, then:

a) Where $* \in \{\cdot, +, \wedge, \vee, \&\}$, $x*x = x$ (idempotency). But when $* \in \{\hat{}, ", \bullet, \not\bullet, \exists\}$, $x*x \neq x$ ($x'x$ is undefined). Special case: when $x = \emptyset$ and $* \in$ $\{\cdot, +, \wedge, \vee, \&, \bullet, \exists\}$, $\emptyset*\emptyset = \emptyset$ (null idempotency).

b) When $* \in \{\cdot, +, \wedge, \vee, \&, \bullet\}$, $x*\emptyset = x* =$ x, and $\emptyset$ is an identity variable. But when $* \in \{\hat{}, ", \not\bullet, \exists\}$, $x*\emptyset = x* \neq x$ ($x'\emptyset$ is undefined).

c) When $* \in \{\hat{}, "\}$, $(x*)* = (x*\emptyset)*\emptyset = x**$ = x (unary closure). When $* \in \{\cdot, +, \hat{}, \vee, "\}$ and x ranges over values only, $(x*)* = x** = x$. When $* \in \{\cdot, +, \hat{}, ', \wedge, \vee, ", \&, \bullet\}$, x ranges over values only, and $x \neq \emptyset$, $(x*)* = x** =$ x. When $* \in \{\not\bullet, \exists\}$, $x** \neq x$. Example: When $r_g = \{0,1,2,3,4\}$ and $x =$ 3, $x++ = 3+ = 3$, $x\cdot\cdot = 3\cdot = 3$, $x\hat{}\hat{} = 1\hat{} = 3$, $x\not\bullet\not\bullet = \emptyset\not\bullet = \{0,1,2,3,4\}$ (i.e., $x\not\bullet\not\bullet \neq x$), $x\exists\exists = 4\exists = 4$ (i.e., $x\exists\exists \neq$ x).

3) $\forall(x,y,z) \subseteq r_g$, $x*(y*z) = (x*y)*z$ where $* \in \{\cdot, +, \wedge, \vee, \&, \exists\}$ (associativity).

4) $\forall(x_i) \subseteq \{r_g \bar{\ } \emptyset\}$ for a given Mx-gate of function $*$ where $* \in \{\cdot, +, \hat{}, \wedge, \vee, ", \bullet, \not\bullet\}$, $x_1 * x_2 * \ldots * x_m * \emptyset_{m+1} * \emptyset_{m+2} *$ $\ldots * \emptyset_n = x_1 * x_2 * \ldots * x_m$. (The subscripted nulls represent inputs each of which is currently null.)

5) Where $x \neq \emptyset$ and r_g is not a single-member set, we observe:
$x \wedge x\hat{} = \emptyset \qquad x \wedge x" = \emptyset \qquad x \vee x" = \{x,x"\}$

2.2.2 Completeness in Mx

Completeness for the Mx GFS and for the F members is conditional[9]. Existing definitions of completeness (e.g., [6,8]) cannot be applied directly to F because they do not treat input alphabets and universes of operation

331

as independent from the operators. Defini-
tions of completeness appropriate to Mx
follow. "Moderate" completeness accompanies
the notions of strong and weak complete-
ness.[4]

Strong Completeness - $F_j \subseteq F$ in Mx is
"strong complete" over a set S of two
or more values iff the minimum count
of iterations of networks (including
the initial one) composed of members
of F_j required to cover S is not less
than the cardinality of S.

Moderate Completeness - $F_j \subseteq F$ in Mx
is "moderate complete" over a set S
having two or more values iff the
minimum count of iterations of net-
works (including the initial one)
composed of F_j members required for
the union of their outputs to be
identical to S is less than the car-
dinality of S.

Citizen - Any input instance which is
a subset of r_g. "Citizen" is the
antonym of "alien."

Weak Completeness - $F_j \subseteq F$ in Mx is
"weak complete" with respect to a set
S where $|S| \geq 2$ iff some citizen(s) of
or alien(s) to the F_j members is(are)
needed at any input(s) at any itera-
tion of the networks in the strong or
moderate complete definitions, ex-
cluding the initial iteration, in
order to make F_j otherwise strong or
moderate complete.
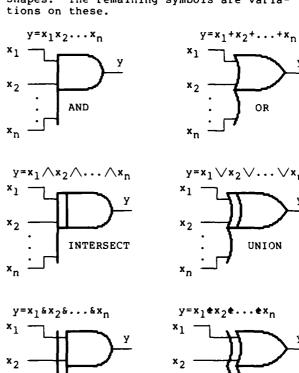
$F_j \subseteq F$ in Mx is moderate complete if
also strong complete. The converse may not
be true. A strong or moderate complete F_j
is weak complete iff all input instances to
its members also belong to S. If F_j is
weak or moderate complete, it cannot be
made stronger without replacing and/or
including one or more functions. Any
strong or moderate complete simple basis[4]
requires the inclusion of at least one
monadic function, or an n-adic function
which gives a defined output with all its
inputs connected together, to service
single-output networks.*
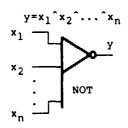
In the following list of predeter-
mined[9] F member completeness, s, m, w, and

---

* A monadic operation is defined as an
operation requiring one input (argument).
An n-ary operation is defined as an m-adic
operation having n inputs, where m≤n.

i mean possibly strong, moderate, or weak
completeness, and incomplete, respectively.
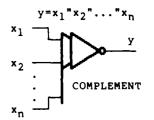
AND	i
OR	i
NOT	w, conditionally s or m
INTERSECT	i
UNION	i
COMPLEMENT	s or m
ANDe	i
ORe	i
NOTe	s or m
EXIST	i

2.2.3 Gate Symbol Graphics for the
Proposed Function Set   Symbols for
AND, OR, and NOT gates are the usual
shapes. The remaining symbols are varia-
tions on these.

$y = x_1 x_2 \cdots x_n$ — AND

$y = x_1 + x_2 + \cdots + x_n$ — OR

$y = x_1 \wedge x_2 \wedge \cdots \wedge x_n$ — INTERSECT

$y = x_1 \vee x_2 \vee \cdots \vee x_n$ — UNION

$y = x_1 \& x_2 \& \cdots \& x_n$ — ANDe

$y = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ — ORe

$y = x_1 \hat{\ } x_2 \hat{\ } \cdots \hat{\ } x_n$ — NOT

$y = x_1 " x_2 " \cdots " x_n$ — COMPLEMENT

$$y = x_1 \neq x_2 \neq \ldots \neq x_n$$

NOTe

$$y = x_1 \exists x_2 \exists \ldots \exists x_n$$

EXIST

When only 2-valued variables exist and over the same set of values, "^" is unary; "'" may be used instead. "'" is undefined for non-unary applications of NOT.

a b	m=ab	n=a+b	p=a^b	q=b^	r=a"b	s=b"	t=a∧b	u=a∨b
0 0	0	0	1	1	0	1	0	0
0 1	0	1	{0,1}	0	∅	0	∅	{0,1}
1 0	0	1	{0,1}	1	∅	1	∅	{0,1}
1 1	1	1	0	0	1	0	1	1
0 ∅	0	0	1	∅	1	{0,1}	0	0
1 ∅	1	1	0	∅	0	{0,1}	1	1
∅ 0	0	0	1	1	1	1	0	0
∅ 1	1	1	0	0	0	0	1	1
∅ ∅	∅	∅	∅	∅	{0,1}	{0,1}	∅	∅
0 2	∅	∅	∅	∅	∅	∅	∅	∅
1 2	∅	∅	∅	∅	∅	∅	∅	∅
∅ 2	∅	∅	∅	∅	∅	∅	∅	∅

2.2.4 Mx Gate Evaluations

This section gives examples of Mx-gate behavior. A treatment of set I/O is included. Variables shown sub- and superscripted with radix and bus signal cardinality, respectively, specify bus realizations.

a b	v=a≠b	w=b≠	x=a&b	y=a●b	z=a∃b
0 0	∅	∅	0	∅	1
0 1	∅	∅	{0,1}	∅	1
1 0	∅	∅	{0,1}	∅	1
1 1	∅	∅	1	∅	1
0 ∅	∅	{0,1}	∅	0	1
1 ∅	∅	{0,1}	∅	1	1
∅ 0	∅	∅	∅	0	1
∅ 1	∅	∅	∅	1	1
∅ ∅	{0,1}	{0,1}	∅	∅	1
0 2	∅	∅	∅	∅	∅
1 2	∅	∅	∅	∅	∅
∅ 2	∅	∅	∅	∅	∅

Example 2.2.4-1

Assume $a \in \{\emptyset, 0, 1\}$ and $b \in \{\emptyset, 0, 1, 2\}$. Note that the input buses are not used to their capacities. The input sets are obvious from the figure. Assume all gate reference sets to be $r_g = \{0,1\}$ with $0 < 1$.

Application of the proposed Mx-function Definitions yields the following truth tables. Note that all gates yield a null when an alien input is present.

Null outputs require physical representation (e.g., a D.C. potential) in practice even though $\emptyset \in$ {no value, empty set, no answer/output, ...}.

If inputs are restricted to be {0,1}, the outputs of the COMPLEMENT and NOT Mx-gates are identical.

For $a \in \{0,1\}$, $b \in \{0,1\}$, and outputs t and u in the truth tables, if the null and set outputs are changed to 0 and 1 respectively, then the operations of AND and INTERSECT, and OR and UNION are indistinguishable.

The truth tables show by exhaustion that AND, OR, and NOT Mx-gates, under 0,1 inputs and {0,1} reference sets, yield results identical to 2-valued Boolean AND, OR, and NOT.

————End of Example 2.2.4-1————

Methods exist for representing and physically handling set I/O.[9] One possibility for outputs is to detect unique set instances within the Mx-gate, then deliver a set identifier to the external circuitry. For example, the symbol "5" can be assigned to a set such as {0,1,2}. A transmitted set identifier (e.g., "5") can be interpreted by receiving Mx-gates as desired.

This set identifier method can be used to "create" elements not in a transmitting gate's reference set. For example, if $r_g =$

{1,2,3}, then outputs of 4 = {1,2,3}, 5 = {1,2}, 6 = {2,3}, 7 = {1,3} can be "created" at its gate's output.
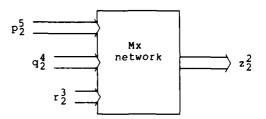
2.3 A Way to Design Using Mx

The following design procedure approximates the truth-table methods familiar to users of 2-valued Boolean algebra.

Procedure 2.3-1 – First-Order Logic Design Using Mx

1) Build a truth table for the problem having columns for its inputs followed by columns for its outputs.

2) Fill each row with an I/O instance so that all possible I/O conditions are covered.

3) For each non-$\emptyset$ input instance, determine (if any) a NOT class gate and reference set to yield it.

4) For each non-$\emptyset$ output, determine an AND class gate and reference set to yield it. If satisfactory gate and reference set pairs are all found, go to step 5. Otherwise, insert a blank column before the output columns.

   a) For each non-$\emptyset$ output, determine an AND class gate and reference set pair and test input (install in the new column) to yield it. If satisfactory input and gate and reference set pair combinations cannot all be found, keep the best ones, insert another blank column before the previous one, then go to step 4a. Otherwise, for each test input given the original inputs, determine an AND class gate and reference set pair to yield it.

5) For each output column and pair or other test tuples of rows having non-$\emptyset$ outputs, determine a solving cascade of OR class gate and reference set pairs. If satisfactory gate and reference set pairs are all found, go to step 6. Otherwise, append a blank column to the truth table.

   a) Similar to step 4a, solve for the test inputs but using OR class gates. Failing, keep the best test sets, insert another blank column before the previous one, then go to step 5a.

6) Draw the circuit and/or write the equations accordingly.

———End of Procedure 2.3-1———

Procedure 2.3-1 does not promise a 2-level AND-OR solution. Especially in steps 3 through 5a, much room for interpretation and style exists, which is why the Procedure is called "First-Order." The following simple example uses Procedure 2.3-1. More substantial examples exist.[9]

Example 2.3-1

Specification: Transmit to output z the value from set {a,b,c} common to buses p, q, and r.

The bus carrying signal p has a physical capacity of $2^5 = 32$ values, an example use of the notations. Output z is physically 4-valued: one for each member of {a,b,c} and one assigned to $\emptyset$. (The super/subscripts have no algebraic significance and make the broad arrow bus symbol unnecessary.)

Steps 1 and 2 of Procedure 2.3-1 yield the following truth table.

p	q	r	z
a	a	a	a
b	b	b	b
c	c	c	c
otherwise			$\emptyset$

Applying step 3, no "inversions" exist (e.g., $a \neq c\hat{}$) for any input instance, so no NOT class gates are needed.

Applying step 4, three non-$\emptyset$ outputs exist, so three AND class gates are needed. Any of an AND, INTERSECT, or ANDe gate yields the value common to their inputs. (Two OR class gates, the OR and UNION, also do this.) Arbitrarily select three INTERSECT gates.
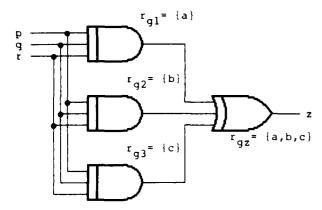
Value "a" is required at the output of the INTERSECT gate assigned to the row where all inputs are equal to "a"; otherwise, $\emptyset$ is. Assigning $r_{g1} = \{a\}$ to the gate accomplishes this. Similarly assign $r_{g2} = \{b\}$ and $r_{g3} = \{c\}$ to the INTERSECT gates for the rows where z=b and z=c, respectively.

By step 5, since no more than one AND class gate output will be non-$\emptyset$ at any time, any single 3-input OR class gate will
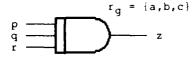
334

suffice. (Three AND class gates, the AND, INTERSECT, and ANDe would also serve as well.) UNION is arbitrarily chosen.

Only the values a, b, or c can appear at the inputs of the chosen UNION gate. Hence, it can have any reference set r_{gz} provided that {a,b,c} is a subset of it. Arbitrarily choose r_{gz}={a,b,c}.

Step 6 yields the following circuit, corresponding to the expression, $z = (p \land q \land r) \lor (p \land q \land r) \lor (p \land q \land r)$.

One is tempted to apply idempotency to the expression for z, hoping to reduce it to $z = p \land q \land r$. Such an application of idempotency is generally invalid, since each gate has a different reference set. However, a single 3-input INTERSECT gate with r_g = {a,b,c} allows $z = p \land q \land r$, reducing the previous circuit to the following equivalent.

Had any AND-class gate other than INTERSECT been chosen, a single-gate solution would not have been possible. All other gates would produce undesired outputs given r_g = {a,b,c}, and no other reference set would have sufficed.

Intuition suggested using a different reference set to arrive at the single-gate equivalent. Procedure 2.3-1 yielded a correct result, but not one having minimum Mx-gate count. Noticing earlier that the truth table describes set-theoretic intersection under {a,b,c,} would have led directly to $z = p \land q \land r$, an application of the definition of INTERSECT.

——End of Example 2.3-1——

2.4 A Way To Turn Mx Designs Into Hardware

Hardware directly realizing the gates of F would be useful.* Two-valued Boolean algebra enjoys that position in small scale ICs (integrated circuits). Some means for realizing Mx-designs have been reported.[9] But one more method, table-lookup,** is simple yet exploits VLSICs (very large scale ICs) for modest size reference sets and input value cardinalities. Table-lookup, applied to the single-gate solution of Example 2.3-1 follows.

Assume the single-gate solution in Example 2.3-1, a 3-input INTERSECT gate, is to be built in a 2-valued technology as indicated by the Example's first diagram. Then, since input r is a 3-wide bus and the remaining two buses are wider, inputs p and q can be reduced to 3-wide buses also.

Assume that the values a, b, and c are represented by the symbols 2, 5, and 7, respectively, in binary arithmetic nota-tion. The value range of interest must include the alphabet {2,5,7}. With that as the only constraint, arbitrarily choose $p \in$ {0-7}, $q \in$ {0-7}, and $r \in$ {0-7}.

Two methods of constructing table-lookup hardware use PLAs (programmable log-ic arrays) or memories. We arbitrarily select an EPROM (erasable, programmable read-only memory).

Since the solution circuit has three 3-wide bus inputs, an EPROM having at least nine (3+3+3) address bits are needed. The output requires at least three bits for non-null representation. Adopting a con-vention where a null output is signified by the most significant bit being a binary one requires that the output be increased to at least four bits. This implementation re-quires two more output bits than suggested in Example 2.3-1 because that version as-sumed the minimal physical configuration. That is, for an n-valued output, no more than $\log_2 n$ signal lines are needed in a bus.

Since nine address bits and four out-put bits are required, an EPROM with 512 4-bit words (a "2K" EPROM) is needed; well within state-of-the-art technology. (Speed is not considered in this implementation.) Therefore, an EPROM programmed as shown in the following truth tables can directly realize the single gate solution to Example 2.3-1. (A generalization of this EPROM method can serve in other Mx-networks as a

---

* A patent has been initiated.

** Idea suggested by Dr. Richard K. Kunze, Dept. of Defense, 24 August

standard realization of an INTERSECT gate. This is known as a "standard cell" approach.[3])

p q r	z	address in decimal notation	output in binary notation
7 7 7	7	511	0111
5 5 5	5	365	0101
2 2 2	2	146	0010
other	∅	other	1xxx

3.0 Conclusions

A technology-independent, heterogeneous, mix-valued logic design algebra denoted by "Mx" was discussed. As an algebra intended for use mainly by logic design practitioners instead of only researchers, Mx is atypical. It has been applied where I/O consists of values and/or sets of values and/or variables. Function I/O appears attractive for preserving numeric representation accuracy.[5] As examples, a) The natural number, e, may be operated on as a token, saving numeric representation until the last moment, and b) Division, even though not in the proposed Mx-function set, may be carried through a network — a fraction such as 1/3 may be symbolized then decimally represented at the last moment, reducing losses in precision due to recursive operations.

The behavior of Mx-gates depends upon their reference sets (which may be different among the Mx-gates) and input alphabets (which may be different among the inputs) for algebraic properties. Reference sets can be varied with time or other parameters. Real-time changes to the nature of an entire hardware circuit may thereby be possible. Many algebraic properties of Mx remain to be explored.

Mx-gate-level minimization methods are as yet unexplored. Their availability could be useful to realization methods like the one discussed in Section 2.4.

Completeness, associativity, and distributivity in the proposed Mx function set are all conditional. Yet by Example 2.3-1 and others[9], Mx has been shown to be useful in the design of logic circuits having 2- or multi-valued signals and their buses.

References

1. BAER, J. L., "A Survey of Some Theoretical Aspects of Multiprocessing," Computing Surveys, vol. 5, No. 1, March 1973, pp. 31-80.

2. ENDERTON, Herbert R., A Mathematical Introduction to Logic, Academic Press, 1972, p. 9.

3. FRANK, Edward H., and SPROULL, Robert F., "Testing and Debugging Custom Integrated Circuits," Computing Surveys, Vol. 13, No. 4, December 1981, pp. 425-451.

4. MUKHOPADHYAY, Amar, "Complete Sets of Logic Primitives," Recent Developments in Switching Theory, Edited by Amar Mukhopadhyay, Academic Press, 1971, pp. 1-26.

5. PETERSON, Ivars, "Can You Count on Your Computer?," Science News, 31 July 1982, Vol. 122, No.5, pp. 72-75.

6. RESCHER, Nicholas, Many-valued Logic, McGraw-Hill Book Company, 1969.

7. RINE, David C., Editor, Computer Science and Multiple-Valued Logic, Theory and Applications, North-Holland, 1977.

8. ROSENBERG, Ivo G., "completeness properties of multiple-valued logic algebras," ibid., pp. 144-186.

9. SINUTKO, Michael Jr., "A Mix-Valued Algebra for Combinatorial Digital Logic Design," Ph.D. Dissertation, University of Maryland, College Park, Maryland, 1982.

A MINIMIZATION METHOD FOR ENGINEERING ESTIMATION

Dr. Suchitra Dhar

Associate Research Scientist
Lockheed Missiles & Space Co.
Sunnyvale, CA 94086

ABSTRACT

This paper obtains a convergence-criterion for optimal estimation by constructing a mathematical theory of ordering, based upon topological and algebraic concepts. This theory provides the model for minimizing the variance of error associated with the estimators of a true state. Thus it is a supplement to the classical Kalman filtering approach. The theory is first described in mathematical terms, as an ordering structure consisting of these entities: a non-empty set of estimators, a binary relation of comparison between estimators, and a closed binary operation that composes the estimators in some prescribed fashion. A triple consisting of these entities is an ordering structure, if and only if the axioms of weak order, associativity, monotonocity, and Archimedean property are satisfied. A weak representation theorem is stated regarding the existence of an order-preserving real-valued function on the set of estimators. A stronger version of that theorem, constructing an actual function for a special case, is demonstrated. The Archimedean convergence of estimators that is different from any numeric computation of variance, is mathematically involved in the Representation Theorem. The algorithm can be implemented by using recent Artificial Intelligence techniques.

Keywords: Algebra, axioms, relations

1. OBJECTIVE

This research aims to shed light on the engineering estimation theory, known as Kalman filtering, by a self-sufficient theory. This new approach is based on qualitative ordering, i.e., on obtaining a binary relation, derived from mathematical properties.

2. BASIC FILTERING

Filtering is done with the model described by the following equations.

$$x_t = \Gamma_{t-1} x_{t-1} + G_{t-1} w_{t-1} \qquad (1)$$

$$z_t = H_t x_t + v_t \qquad (2)$$

where x is the n-dimensional state-vector, F is the nxn state transition matrix, G is the nxp noise matrix, w is the p-dimensional noise vector, z is the m-dimensional measurement vector.

H is the mxn measurement matrix and v is the m-dimensional distortion on the measurement.

The problem of filtering is to find recursively the best estimator x_t such that the variance of

$(x_t - \hat{x}_t)$ is minimized. Estimation theory has a vast literature. See, for example, references [1], [2] and [3]. The following is a brief description of filtering.

There is a true physical state, e.g., the true position and velocity of a satellite in orbit. We want to estimate chat true state as accurately as possible. In order to do so, the following model is assumed. The state x_t is a random variable that changes dynamically according to the above matrix differential equation. The state itself is disturbed by noise w. G is the matrix that determines how the noise is distributed.

It is assumed in classical Kalman filtering theory that the measurements z are given as linear combinations of the elements of the state-vector $H_t x_t$ plus some measurement noise v_t. Noises cause the measurement error.

The sense of approximation of the true state is taken to be the least-squares criterion. That is, the variance of $x_t - \hat{x}_t$ is minimized. The solutions of these equations can be implemented efficiently in the computer. This is done recursively, that is, from the estimate of the previous times, by updating the estimate from the previous time to the present time. [1] [2] [3]

3. THE BASIC CONCEPT OF MINIMIZATION OF THE PRESENT MODEL

The concept of minimization is to choose the best estimator of a true state, among several candidates, $\hat{x}, \hat{y}, \hat{z}$, by comparing them two at a time with a representation of qualitative ordering between them. So the conventional minimization of error-variance is transformed into a relational form of ordering of any two estimators. The ordering of two variables regarding their variances is now considered.

4. FORMULATION OF THE THEORY

4.1. Elements

The theory is formulated in terms of the set A,

consisting of pairs of estimators and their weights. There are three primitives--a nonempty set A, a binary relation of comparison $\gtrsim$ on A, and a closed binary operation o that maps AxA into A. The interpretation is that A is a set of entities that exhibit the attribute in question (in our case, variance). $\hat{x} \gtrsim \hat{y}$ holds if and only if $\hat{x}$ exhibits, in some prescribed qualitative way, at least as much of the attribute a $\hat{y}$ does, and $\hat{x}$ o $\hat{y}$ is an entity in A that is obtained by composing $\hat{x}$ and $\hat{y}$ is some specific way. For the sake of notational convenience, from now on we shall use x and y for the estimators $\hat{x}$ and $\hat{y}$.

4.2. Definitions

Let A be a nonempty set of estimators, $\gtrsim$ a binary relation, on A, and o a closed binary operation on AxA. The triple $\langle A, \gtrsim, o\rangle$ is an extensive structure iff the following four axioms are satisfied for all elements in A.

(1) Weak order: $\langle A, \gtrsim\rangle$ is a weak order. That is, $\geq$ is a reflexive, transitive and connected relation.
(2) Associativity: a o (b o c)$\sim$(a o b) o c.
(3) Monotonocity: a $\gtrsim$ b iff a o c $\gtrsim$ b o c iff c o a $\gtrsim$ c o b, for some c in A.
(4) Archimedean: If a $>$ b, then for any c,d in A, there exists a positive integer n such that na o c $\gtrsim$ nb o d, where na is defined inductively as : la=a, (n+1)a = na o a.

5. EXAMPLE OF ORDERING BY AXIOMS

A special case of estimator-weights pairs, where the weight is interpreted as the inverse of the variance (σ^2) of the scalar estimator, is presented next in order to show that the qualitative axioms of the definition in Sec.4.2 can be verified.

5.1. Weighted Least Squares Estimate

As shown in Sec.2, filtering is a form of recursive least squares estimate. This section briefly explains that least squares method, yielding the concept of weights.

The linear least squares problem involves using a set of measurements z, which are linearly related to the unknown quantities x by the expression

$$z=Hx + v \qquad (3)$$

where v is a vector of measurement noise. The goal is to find an estimate of the unknown, denoted by x,y or z. Given the vector difference z-Hx, we wish to find the x that minimizes the sum of the squares of the elements of z-Hx. The vector inner product generates the sum of the squares of a vector. Thus, the scalar cost function J is minimized where

$$J=(z-Hx)^T(z-Hx) \qquad (4)$$

Minimization of a scalar, with respect to a vector, is obtained when

$$\frac{\partial J}{\partial x} = 0 \qquad (5)$$

and the Hessian of J is positive semidefinite

$$\left|\frac{\partial^2 J}{\partial x^2}\right| \gtrsim 0. \qquad (6)$$

Differentiating J and setting the result to zero yields

$$H^T Hx = H^T z \qquad (7)$$

The second derivative of J, with respect to x, is positive semidefinite; and thus the equation (7) does, indeed, define a minimum. When $H^T H$ possesses an inverse, the least squares estimate is

$$x = (H^T H)^{-1} H^T z \qquad (8)$$

The above least-squares derivation is based on the assumption that all measurements z are of equal quality. If, in fact, it is known that it is reasonable to apply different weights to the various measurements comprising z, the least squares estimator should be appropriately modified. If the ith measurement z_i has a relative weight of W_i, it is reasonable to construct a diagonal matrix W with $W_1^2, W_2^2, ..., W_n^2$ on the diagonal. In that case, the least squares estimate is

$$x = (H^T WH)^{-1} H^T Wz \qquad (9)$$

For the special case under consideration, weights are taken to be inverse of the variance σ^2, estimators are scalar, o (the closed binary operation on A) is interpreted as weighted average of estimators, and the binary relation $\gtrsim$ is interpreted as minimizing the variance of error associated with each estimator.

Given the above interpretation, the following calculations show how the closed operation yields a new estimator.

$$z= \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \overset{H}{\begin{bmatrix} 1 \\ 1 \end{bmatrix}} x + \overset{V}{\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}} \qquad (10)$$

v_1 is $N(0,\sigma^2)$

v_2 is $N(0,\sigma^2)$

$$x_3 = (H^T WH)^{-1} H^T W_z$$

$$= \left(\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} W_1 & 0 \\ 0 & W_2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} W_1 & 0 \\ 0 & W_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= (W_1 + W_2)^{-1} \begin{bmatrix} W_1 & W_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= \frac{W_1 x_1 + W_2 x_2}{W_1 + W_2} = \frac{W_1}{W_1 + W_2} x_1 + \frac{W_2}{W_1 + W_2} x_2 = 1$$

$$\sigma_3^2 = \left[\frac{W_1}{W_1 + W_x} \sigma_1^2 + \frac{W_2}{W_1 + W_2} \sigma_2^2 \right]^{\frac{1}{2}} \qquad (11)$$

The W_3 is derived in the following way:

$$W_3 = \frac{1}{\sigma_3^2} = \frac{1}{\left(\frac{W_1}{W_1+W_2}\right)^2 \cdot \frac{1}{W_1} + \left(\frac{W_2}{W_1+W_2}\right)^2 \cdot \frac{1}{W_2}}$$

$$= \frac{1}{\dfrac{W_1}{(W_1+W_2)^2} + \dfrac{W_2}{(W_1+W_2)^2}} = W_1+W_2 \qquad (12)$$

Hence, a third element of the set A, where the elements are estimator-weight pairs, is derived by the concatenation of two other elements in the following way:

$$(x_1,W_1) \; o \; (x_2,W_2) = \left(\frac{W_1 x_1 + W_2 x_2}{W_1 + W_2} \; , \; W_1+W_2 \right) \qquad (13)$$

5.2. Verification of the Axioms

Using the special case of estimator-weight pairs, the following elementary calculations show that the axioms are verified in that case.

(1) Weak order. This property is trivially verified in the case of estimator-weights pairs, since weights are real numbers, and all real numbers can be weakly ordered, i.e., they are reflexive, transitive, and connected.

(2) Associativity. The following elementary calculations with the estimator-weight pairs will show that both sides are qualitatively equivalent, i.e., $\sim$ holds. Let us adopt the convention that the estimators be called a,b,c---and their respective weights be called W_a, W_b, W_c, etc. Then the Associativity axiom takes the following form:

$$(a,W_a) \; o \; \Big((b,W_b) \; o \; (c,W_c) \Big)$$
$$\sim \Big((a,W_a) \; o \; (b,W_b) \Big) \; o \; (c,W_c) \qquad (14)$$

First, we note the results of concatenation on the left-hand side.

$$(b,W_b) \; o \; (c,W_c) = \left(\frac{W_b b + W_c c}{W_b+W_c} \; , \; W_b + W_c \right)$$

$$(a,W_a) \; o \; \left(\frac{W_b b + W_c c}{W_b+W_c} \; , \; W_b + W_c \right) =$$

$$\left(\frac{W_a a + W_b b + W_c c}{W_a+W_b+W_c} \; , \; W_a + W_b + W_c \right) \qquad (15)$$

Next, we check the results on the other side to see if they are equivalent.

$$(a,W_a) \; o \; (b,W_b) = \left(\frac{W_a a + W_b b}{W_a + W_b} \; , \; W_a + W_b \right)$$

$$\left(\frac{W_a a + W_b b}{W_a+W_b} \; , \; W_a + W_b \right) \; o \; (c,W_c)$$

$$= \left(\frac{W_a a + W_b b + W_c c}{W_a + W_b + W_c} \; , \; W_a + W_b + W_c \right) \qquad (16)$$

Therefore, associativity holds.

(3) Monotonocity. It is noted from sec.5.1 that when two elements of the set A are concatenated, their weights are added. In the terms of estimator-weight pairs, montonocity is verified by noting that

$$(a,W_a) \; o \; (c,W_c) = \left(\frac{a\,W_a + c\,W_c}{W_a + W_c} \; , \; W_a + W_c \right),$$

and $(b,W_b) \; o \; (c,W_c)$ also yields the weight W_b+W_c so, $(a,W_a) \succcurlyeq (b,W_b)$ iff $(a,W_a) \; o \; (c,W_c)$

$$\succcurlyeq (b,W_b) \; o \; (c,W_c) \qquad (17)$$

This is shown to hold by the fact that

$$W_a + W_c > W_b + W_c \text{ iff } W_a > W_b \qquad (18)$$

This certainly (and trivially) holds for real numbers.

(4) Archimedean. Before we verify this property, let us adopt some conventions.

Let $(a,W_a) \succcurlyeq (b,W_b)$ mean that a is better than b. In this case, σ^2 of a $\leq \sigma^2$ of b. That is, $\frac{1}{W_a} \leq \frac{1}{W_b}$.

Since these quantities are always positive, one can take reciprocals.

$$W_a \succcurlyeq W_b$$

Therefore,

$$(a,W_a) \succcurlyeq (b,W_b) \iff W_a \succcurlyeq W_b \qquad (19)$$

The above is our hypothesis.

According to the formulation of the Archimedean property, there is no hypothesis about (c,W_c) and (d,W_d) except that they are also estimator-weight pairs. The conclusion that we want to draw is that there exists an n such that

$$n(a,W_a) \; o \; (c,W_c) \succcurlyeq n(b,W_b) \; o \; (d,W_d) \qquad (20)$$

By elementary calculations, it can be shown that

$$n(a,W_a) = (a, n\,W_a) \qquad (21)$$

So, $(a, n\,W_a) \; o \; (c,W_c) \succcurlyeq (b, n\,W_b) \; o \; (d,W_d) \qquad (22)$

Now we concatenate the left-hand side of equation (22).

$$\left(\frac{n(W_a)\,a + W_c\,c}{n(W_a) + W_c} \; , \; n(W_a) + W_c \right) \qquad (23)$$

and also the right-hand side,

$$\left(\frac{n(W_b)\,b + W_d\,d}{n(W_b) + W_d} \; , \; n(W_b) + W_d \right) \qquad (24)$$

Next we have to prove that there exists an n such that the weight of the left-hand side is quantitatively greater than the weight of the right-hand side, given the hypothesis $W_a > W_b$.

The only facts about numbers that we need to prove this is that the real numbers form an Archimedean system, and the weights are real numbers. So, the Archimedean property of weighted estimation is a direct (and trivial) consequence of the Archimedean property of real numbers.

6. CONSEQUENCES OF THE AXIOMS

6.1. Representation Theorem

A somewhat weak assertion is made by the following representation theorem, which follows from the axioms in general. Several versions of this theorem can be found in Krantz, Luce, Suppes, Tversky [4]. The following is a formulation: $\langle A, \succcurlyeq, o \rangle$ is an extensive structure iff there exists a real-valued function R on A ($R: A \rightarrow \mathbb{R}$) such that for

all x,y in A

(i)$x \succsim y$ iff $R(x) \geqslant R(y)$.

(ii)$R(x \circ y) = R(x) + R(y)$.

That is, R is a homomorphism from $\langle A, \succsim, \circ \rangle$ onto an ordered additive semigroup of reals in which $\succsim$ is the preimage of $\geqslant$.

The Archimedean property has been formulated in terms of multiple-valued logic (Katz [5] [6]), while we use general topology and algebra. The current paper is a thoroughly revised version of Dhar [7].

6.2. Stronger Consequence of the Axioms

The Representation Theorem, as stated above, is trivial in the particular case of weights of scalar estimators. We have achieved more than what the theorem states--we have explicitly constructed the real-valued function R, where R is the weight of the estimator. Note that the verification of the axioms have shown that the verified axioms imply the existence and construction of the R, and not vice versa.

6.3. Archimedean Convergence

The general version of the Archimedean convergence, found in [4], is applicable in this case too. A rough sketch of the concept of convergence is as follows: Select any x in A, as the unit. For any other y in A, and for any positive integer n, the Archimedean axiom guarantees that there is an integer, for which $mx \succ ny$. Let m_n be the least integer for which this is true. As n is selected to be larger and larger, the approximation gets closer and closer, and, if the limit exists, we define
$R(y) = \lim_{n \to \infty} m_n/n$

7. ALGORITHM

The following is an algorithm for obtaining the desired results:
1) Take pairs of estimator-weights (assumed to be yielded by previous calculations) as elements of the set A. Note that the weight of an estimator does not have to be inverse variance; it can be interpreted in many other ways, such as standard deviation.
Another way of considering weights is to understand that the same random variable can be minimized by linear or nonlinear estimators, thus yielding different concepts of weights. For example, a linear estimator (of, say, a quadratic random variable) is a linear function of observations. The errors that are estimated by such an estimator are penalized linearly; the weights are the same whether the error is large or small. On the other hand, if the same random variable is estimated by a nonlinear estimator which is a nonlinear function of observations, then small errors are weighted differently than large errors. Such considerations lead to the formation of different optimization criteria. More about this will be discussed in sec. 9.

2) Whatever the interpretation of weight is, each of the qualitative axioms is verified by using LISP-based program and Artificial Intelligence

techniques.

3) If the pairs satisfy the axioms, they yield a "winner", the optimal estimator, after comparing the pairs two at a time. Note that this comparison does not require prior knowledge of the ordering between any two estimators. Assume that the ordering holds one particular way, and the satisfaction of the axioms will show whether that particular way is correct or not.

4) Once the optimal estimator is determined, check for convergence.

8. POSSIBLE OBJECTION TO THE ALGORITHM

One immediately anticipated objection to this algorithm is that a closed extensive structure generates an infinite number of elements. That can be countered by selecting estimators according to some of their properties derived from the Kalman filtering procedure itself. More about this process will be expounded in the later development of this work.

9. A VARIATION ON THE ALGORITHM

We noted that the above algorithm does not depend on the ordering to be known prior to embarking on the process. The ordering is determined by the LISP-processing of qualitative properties. This determination and convergence are two primary outcomes of the algorithm. But the algorithm can be used, with some variation, for other purposes too. For example, take the case where one is interested in estimating the scalar random variable x^2, given x (with zero mean). In this case, clearly, the best non-linear estimator under any criterion will be $\hat{x}^2$. One may still want to know the nature of a linear least-squares estimator.

$$\widehat{(x^2)} = \left(\frac{Ex^3}{Ex^2}\right) x + Ex^2 \qquad (25)$$

where $\alpha = \dfrac{Ex^3}{Ex^2}$, $\beta = Ex^2$

$$\therefore \widehat{(x^2)} = \alpha x + \beta \qquad (26)$$

This can be proved by using 1) orthogonality
$$\left(x^2 - \widehat{(x^2)}\right) \perp x \qquad (27)$$

and 2) unbiassedness
$$E\,\widehat{(x^2)} = Ex^2 \qquad (29)$$

So, one knows the prior ordering in this case: $\hat{x}^2$ is superior to $\hat{x}$. One may still run the above algorithm, in order to figure out by the representation, how much the measure of one is better than that of the other. But, one can easily run into difficulty. The connectedness property of the weak order axiom demands that any two elements of the set A can be ordered. If only one ordering between two given elements of the set are known, then how is one to order, say, x^4 and x^8, which may also be members of the same set? One way of avoiding this difficulty is to introduce a new step in the algorithm, namely, to perform some calculations on the estimators to bring them as close to the initial ordering as possible, so that the winner that is

known from the beginning, $(\hat{x}^2)$ can still be ordered with any others, and thus make all other orderings known.

10. RELATION TO MULTIPLE-VALUED LOGIC

We noted in sec. 6.1 that the Archimedean property has been formulated by Katz [5][6], in terms of multiple-valued logic. Since the step 2) of the algorithm presented in sec. 7 is the computerized verification of axioms, an extension of such a program will also enable one to obtain the consequences related to multiple-valued logic.

Acknowledgments

References

[1] Gelb, A., Applied Optimal Estimation, The M.I.T. Press, Cambridge, Mass., 1974.

[2] Jazwinski, A.H., Stochastic Processes and Filtering Theory, Academic Press, New York, 1970.

[3] Kailath, T., Lectures on Wiener and Kalman Filtering, Springer-Verlag, 1981.

[4] Krantz, D.H., Luce, R.D., Suppes, P., Tversky, A., Foundations of Measurement, Vol. 1, Academic Press, New York, 1971.

[5] Katz, M., "Two Systems of Multi-valued Logic for Science," pp. 175-182, Proceedings of the Eleventh International Symposium on Multiple-valued Logic, New York: IEEE Computer Society Press, 1981.

[6] Katz, M., "Measures of Proximity and Dominance," pp. 370-377, Proceedings of the Second World Conference on Mathematics at the Service of Man, Spain, 1982.

[7] Dhar, S., "An Order-Theoretic Minimization Method: A Supplement to Filtering Approach," pp. 246-250, Proceedings of the Second World Conference on Mathematics at the Service of Man, Spain, 1982.

THE OPTIMIZATION OF GMC OVER GF(p)

by Ping Dong

Department of Information Engineering
Northwest Telecommunication Engineering Institute
Xi'an, People's Republic of China

Abstract

This paper deals with the obtaining and optimization of the Generalized Reed-Muller Canonical form (GMC) of p-valued logical functions. The basic ideas involved are the linear transforms of logical functions over GF(p) and the processing of the spectra thus obtained. The basic concern is with the convenience of the calculation rather than the analyticity of the process, so that they can be easily implemented using a digital computer.

Introduction

Let $f(x_0,x_1,\ldots,x_{n-1})$ be a p-valued logical function with n arguments, where $x_i \in \{0,1,\ldots,p-1\}$ and p be a prime. It can be regarded as an element of ring $F[\dot{x}_0,\dot{x}_1,\ldots,\dot{x}_{n-1}]$ over GF(p), i.e.

$$f(x_0,x_1,\ldots,x_{n-1}) = \sum_{w=0}^{p^n-1} s(w)\dot{x}_0^{w_0}\dot{x}_1^{w_1}\ldots\dot{x}_{n-1}^{w_{n-1}}$$
$$(\bmod\ p) \qquad (1)$$

where $s(w) \in \{0,1\}$, $\dot{x}_k = x_k + p_k$, $p_k \in GF(p)$ and w_k the k'th element of w in p-ary number expansion(k=0,...,n-1).

For a definite set of p_k, Eq.(1) is unique for every p-valued function $f(x_0,x_1,\ldots,x_{n-1})$[1], and is called the Generalized Reed-Muller Canonical form (GMC) of $f(x_0,x_1,\ldots,x_{n-1})$ and the Reed-Muller Canonical form (RMC) for p=2.

These form of logical functions find important applications in the analysis and synthesis of logical functions, and the error detection of logical networks[2,3]. Here two problems are involved:

1) Given a p-valued logical function (usually its truth table), how to find its GMC expansion coefficients s(w);
2) The coefficients s(w) would be different for different set of p_k(k=0,...,n-1). Thus, for the given function $f(x_0,x_1,\ldots,x_{n-1})$, how to find the optimized set of p_k(the optimized polarity for p=2), so that the expansion is optimal under given criteria.

These two problems have been discussed in previous publications[4]-[10]. In the present paper, they are studied by using the linear transforms and processing the spectral coefficients obtained from these transforms. For the reason of simplicity, let us start with p=2, i.e. the case of the Boolean functions.

Linear Transform over GF(2) and Its Fast Algorithm

In the 2^n dimension linear space over GF(2), a linear transform matrix can be defined as follows:

$$G_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} ;\ G_n = G_1 \times G_{n-1},\ n=2,3,\ldots \qquad (2)$$

where "X" is Kronecker product which is defined as:

$$A \times B = (a_{ij}B)$$

Therefore, Eq.(2) can be rewritten as:

$$G_0 = 1;\ G_n = \begin{bmatrix} G_{n-1} & 0 \\ G_{n-1} & G_{n-1} \end{bmatrix},\ n=1,2,\ldots \qquad (3)$$

It is easy to see that G_n (n ≥ 1) is a triangular matrix with diagonal elements "1". Therefore G_n is nonsingular. In addition, because

$$G_1 G_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

therefore $G_1^{-1} = G_1$. Suppose $G_k^{-1} = G_k$, then we have

$$G_{k+1}G_{k+1} = \begin{bmatrix} G_k & 0 \\ G_k & G_k \end{bmatrix}\begin{bmatrix} G_k & 0 \\ G_k & G_k \end{bmatrix}$$

$$= \begin{bmatrix} G_kG_k & 0 \\ 0 & G_kG_k \end{bmatrix} = \begin{bmatrix} I_{2^k} & 0 \\ 0 & I_{2^k} \end{bmatrix} = I_{2^{k+1}}$$

where I_{2^k} and $I_{2^{k+1}}$ are identity matrices of order 2^k and 2^{k+1} respectively. So we see that

$$G_{k+1}^{-1} = G_{k+1}$$

A conclusion can be drawn from the above that for any n ≥ 1, $G_n^{-1} = G_n$.

For any given vector F in the 2^n dimension linear space over GF(2), a linear transform pair can be defined as follows:

$$\begin{cases} S = G_n^{-1}F = G_n F \\ F = G_n S \end{cases} \qquad (4)$$

Hereafter we shall call S in Eq.(4) the spectral vector of F under the linear transform G_n.

From Eq.(2), we at once get a fast recurrence algorithm of this linear transform:

Algorithm A:

2^n dimension algorithm flow graph

where $\diagdown$ denotes modulo-2 addition.

In addition to this, it is well known that for a matrix obtained from Kronecker product, a fast algorithm always exists[11]. By the decomposition of G_n, an alternative fast algorithm can be obtained.

Algorithm A':

$$a_0(x) = f(x);$$
$$\begin{cases} a_i(t) = a_{i-1}(2t) \\ a_i(t+2^{n-1}) = a_{i-1}(2t) \oplus a_{i-1}(2t+1); \end{cases} \quad (5)$$
$$s(w) = a_n(w)$$

where $i = 1,2,\ldots,n;\ t = 0,1,\ldots,2^{n-1}-1;$
$x,w = 0,1,\ldots,2^n-1$

Now we are going to derive the analytic form of Eq. (4). Firstly, we note that for the w'th element of S in Eq. (4), we have

$$s(w) = (w^0,w^1,\ldots,w^{2^n-1})\begin{bmatrix} f(0) \\ f(1) \\ \vdots \\ f(2^n-1) \end{bmatrix} = \sum_{x=0}^{2^n-1} f(x)w^x$$

$$\text{(mod 2)} \qquad (6)$$

$w = 0,1,\ldots,2^n-1$

where $w^x = w_0^{x_0}w_1^{x_1}\ldots w_{n-1}^{x_{n-1}}$,
and

$$w_k^{x_k} = \begin{cases} 1, & \text{for } x_k = 0 \\ x_k, & \text{for } x_k = 1 \end{cases} \quad k = 0,1,\ldots,n-1.$$

Similarly, for the x'th element of F we have

$$f(x) = \sum_{w=0}^{2^n-1} s(w)x^w \qquad \text{(mod 2)} \qquad (6')$$

$x = 0,1,\ldots,2^n-1$

here the definition of x^w is similar to that of w^x.

For any logical function $f(x_0,x_1,\ldots,x_{n-1})$ of n arguments, we can properly arrange its values to form a vector in 2^n dimension linear space of GF(2)

$$(f(0),f(1),\ldots,f(x),\ldots,f(2^n-1))^T$$

where

$$x = \sum_{k=0}^{n-1} 2^{n-1-k}x_k$$

Now we note that the expression $f(x)$ in Eq. (6) is just the RMC expansion Eq.(1) of $f(x_0,\ldots,x_{n-1})$ for p=2. And the linear transform pair of Eq. (4) enables us to find rapidly the RMC expansion coefficients for any given logical function.

The Optimization Procedure

We know that for any $p_0 \in \{0,1,\ldots,2^n-1\}$ and for any logical function $f(x)$ (hereafter we shall use $f(x)$ instead of $f(x_0,\ldots,x_{n-1})$ for the reason of simplicity), we always have

$$f(x) = f(x \oplus p_0 \oplus p_0) \qquad (7)$$

where "$\oplus$" means diadic addition.

The optimization can be achieved by the following two steps:
1) Find a optimal polarity vector p_0, such that the expansion of $f(x \oplus p_0)$ is optimal under given criterion;
2) Polarize the arguments of $f(x)$ according to p_0. Since for the k'th component of p_0

if $p_{0k} = 0$ then $x_k \longrightarrow x_k = x_k \oplus p_{0k}$

if $p_{0k} = 1$ then $x_k \longrightarrow \bar{x}_k = x_k \oplus p_{0k}$

$k = 0,1,\ldots,n-1.$

This polarization of arguments will be equivalent to diadic addition of p_0 and x, and by Eq.(7), we obtain the optimal RMC expansion of $f(x)$.

Firstly we consider the case $|p| = 1$. Without loss of generality, we let $p_k = 1$ and $p_l = 0\ (l \neq k)$. We note that for any $a,b \in \{0,1\}$

$$1) \quad a^{1 \oplus b} = \begin{cases} a, & b=0 \\ 1, & b=1 \end{cases} = \begin{cases} 1 \oplus \bar{a}, & b=0 \\ a \oplus \bar{a}, & b=1 \end{cases} = a^b \oplus \bar{a};$$

$$2) \quad b = bb^a.$$

Therefore

$$w^{x \oplus p} = w_0^{x_0}w_1^{x_1}\ldots w_{k-1}^{x_{k-1}}w_k^{1 \oplus x_k}w_{k+1}^{x_{k+1}}\ldots w_{n-1}^{x_{n-1}}$$

$$= w_0^{x_0}w_1^{x_1}\ldots w_{k-1}^{x_{k-1}}(w_k^{x_k} \oplus \bar{w}_k)w_{k+1}^{x_{k+1}}\ldots w_{n-1}^{x_{n-1}}$$

$$= w^x \oplus w_0^{x_0}\ldots w_{k-1}^{x_{k-1}}\bar{w}_k \bar{w}_k^{x_k}w_{k+1}^{x_{k+1}}\ldots w_{n-1}^{x_{n-1}}$$

$$= w^x \oplus \bar{w}_k(w \oplus p)^x$$

So that, by Eq. (6) we have

$$s_p(w) = \sum_{x=0}^{2^n-1} f(x \oplus p)w^x = \sum_{x=0}^{2^n-1} f(x)w^{x \oplus p} \quad (\text{mod } 2)$$

$$= \sum_{x=0}^{2^n-1} f(x)(w^x \oplus \bar{w}_k (w \oplus p)^x) \quad (\text{mod } 2)$$

$$= \sum_{x=0}^{2^n-1} f(x)w^x \oplus \bar{w}_k \sum_{x=0}^{2^n-1} f(x)(w \oplus p)^x \quad (\text{mod } 2)$$

$$= s(w) \oplus \bar{w}_k s(w \oplus p) \quad (8)$$

We note that $s(w \oplus p)$ is only a diadic translation of $s(w)$ and that $|p| = 1$. Also we note that in $[0, 2^n)$ w_k takes 0 or 1 alternatively with a length 2^{n-1-k}. We divide interval $[0, 2^n)$ into 2^{k+1} subintervals with the same length 2^{n-1-k} which are denoted by $I_1, I_2, \ldots, I_{2k+1}$. Look at the subinterval I_1. If l is odd, then $w_k = 0$ over this subinterval, therefore $\bar{w}_k = 1$. By Eq. (8)

$$s_p(w)\big|_{w \in I_1} = s(w)\big|_{w \in I_1} \oplus s(w \oplus p)\big|_{w \in I_1}$$

$$= s(w)\big|_{w \in I_1} \oplus s(w)\big|_{w \in I_{l+1}}$$

If l is even, then $w_k = 1$ over this subinterval, therefore $\bar{w}_k = 0$. So that

$$s_p(w)\big|_{w \in I_1} = s(w)\big|_{w \in I_1}$$

Now we obtain a basic algorithm for finding $s_p(w)$ from $s(w)$ when $|p| = 1$:

Algorithm B: When the k'th bit of p is 1 and the others are 0 ($k = 0, 1, \ldots, n-1$), divide the interval $[0, 2^n)$ of w into 2^{k+1} equal sections which are denoted respectively by $I_1, I_2, \ldots, I_{2k+1}$. To obtain $s_p(w)$ from $s(w)$, we need only to add each section with even subscript to its previous section with odd subscript.

For any given p, let $|p_1| = 1$, $p' = p \oplus p_1$. By Eq.(8) we have

$$s_p(w) = s_{p'}(w) \oplus \bar{w}_k s_{p'}(w \oplus p_1)$$

Therefore, once we obtain the RMC coefficients of polarity p, the RMC coefficients of polarity p' with $|p' \oplus p| = 1$ can be easily obtained by algorithm B. Now we have the following optimization procedure:

1). For the given function F, compute its RMC coefficients S by algorithm A or A';
2). Arrange polarity vectors 1 to 2^n-1 in Gray code, and find RMC coefficients for all the polarity vectors by algorithm B;
3). According to the given criterion (criteria), find out the optimal coefficients from all those obtained from 2). The corresponding expansion of $f(x \oplus p_0)$ can be written out according to Eq.(6'), and at the same time we get p_0;
4). Polarize the arguments in the obtained expansion according to p_0, and this gives us the optimal RMC expansion of $f(\bar{x})$.

Optimization Procedure for Some Special Classes of Boolean Functions

Here the basic thought is to reduce the searching range of the optimal polarity vector p_0. We now choose the optimal criterion to be the minimum of the RMC expansion terms.

Linear Functions and Complements of Linear Functions

A logical function is said to be a linear function (or the complement of a linear function), if and only if there exist numbers $c_i \in \{0, 1\}$; $i = 0, \ldots, n-1$, such that

$$f(x) = \sum_{i=0}^{n-1} c_i x_i \quad (\text{mod } 2) \quad (9)$$

$$(\text{or } f(x) = 1 \oplus \sum_{i=0}^{n-1} c_i x_i \quad (\text{mod } 2)) \quad (9')$$

Obviously for a linear function $f(x)$, Eq.(9) itself is one of the optimal RMC expansion. It is easy to verify that the complement of any even number of the arguments which appear in Eq.(9) (i.e. $c_i \neq 0$) is an optimal polarization. For the same reason, the complement of any odd number of the arguments in Eq.(9') is an optimal polarization for a complement of linear function.

Partially Self-dual and Partially Anti-self-dual Functions

A Boolean function is said to be partially self-dual (or partially anti-self-dual) if and only if there exists $x_0 \in \{0, \ldots, 2^n-1\}$, such that

$$f(x) = \overline{f(x \oplus x_0)} \quad (10)$$

$$(\text{or } f(x) = f(x \oplus x_0)) \quad (10')$$

And x_0 here is called a self-duality point (anti-self-duality point) for $f(x)$.

Karpovsky[12] pointed out that the set of all anti-self-duality points for any Boolean function is an Abelian group, and the number of such points (the order of the group) is a power of 2. We can prove that (see Appendix) the union of the all anti-self-duality point set B and the all self-duality point set A is an Abelian group, so that its order is also a power of 2.

We note that if we take a polarity vector p_a from the set A of anti-self-duality points, the RMC coefficients are invariant (because in this case, we have $f(x) = f(x \oplus p_a)$), and if we take a polarity vector p_b from the set B of self-duality points, since $f(x \oplus p_b) = 1 \oplus f(x)$ and by Eq.(2), the spectrum for a constant 1 is $\delta(w)$ ($\delta(w) = 1$ for $w = 0$ and $\delta(w) = 0$ for $w \neq 0$). By the linearity of the transform, we have: If p_b is a self-duality point of $f(x)$, then we have its spectrum

$$s_{p_b}(w) = \delta(w) \oplus s(w)$$

i.e. the first element of $s_{p_b}(w)$ is a complement of $s(w)$ and the others invariant. So, if $f(x)$ has self-duality(anti-self-duality) points, we have the following optimization procedure:

1) Decompose the group P of all polarity vectors into a direct sum of two subgroups H and V with V = A⊕B(This can always be done for the Abelian group P with operator ⊕):

$$P = H \oplus V$$

2) Select polarity vectors from H and find the optimal one which gives the minimum weight of S. Denote this polarity vector by p_0^*;

3) If $s_{p_0^*}(0) = 0$, then $p_0^* \oplus B$ will be the set of optimal polarity vectors. Otherwise $p_0^* \oplus A$ will be the set of optimal polarity vectors and $s_{p_0^*}(0)$ be set to 0.

If $|V| = 2^m$, the above procedure will reduce the amount of operation to $1/2^m$ of the original procedure. An extreme condition is when m = n, this is the case of linear functions.

It can be proved that if the weight $\|f\|$ of the given function is an odd number, then no self-duality(or anti-self-duality) point will exist (except x = 0). It is easy to see that $\|f\|$ be odd if and only if $s(2^n-1) = 1$. In this case, if we use $\delta(x) \oplus f(x)$ instead of $f(x)$ and note that the corresponding spectrum will become $1 \oplus s(w)$, then the above procedure is still applicable.

Partially Symmetric Functions

A Boolean function is said to be partially symmetric if and only if there exist x_i and $x_j(i,j = 0,\ldots,n-1)$, such that for any set of $x_0x_1\ldots x_{n-1}$

$$f(\ldots x_i \ldots x_j \ldots) = f(\ldots x_j \ldots x_i \ldots) \qquad (11)$$

If the above relation is true for every pair of x_i and x_j, then $f(x)$ is symmetric.

When a function is partially symmetric with respect to x_i and x_j, we need only consider the combination 00, 01, 11 or 00, 10, 11 of x_i and x_j. The function will remain unchanged when we substitute 1,0 for 0,1 into x_i,x_j. So that we need only to find out $(2+1)2^{n-2}$ spectra instead of 2^n. Generally speaking, if $f(x)$ is symmetric with respect to k arguments, we need only to find out $(k+1)2^{n-k}$ spectra. When k = n(i.e. $f(x)$ is symmetric), we need to find out n+1 RMC's, this agrees with the result in [13].

We note that when the k symmetric arguments are removed, previous optimization procedure is still valid for the left n-k arguments. The optimization procedure for partially symmetric functions is as follows:
1) Arrange the polarity vectors corresponding to the left n-k arguments in Gray code;
2) Let one of the k arguments be 1 and the others be 0; two of those be 1 and the others be 0; ...etc., each time we get 2^{n-k} spectra. Find out the spectrum which has minimum weight, and this will be the desired RMC expansion coefficients.

At the end of this section, it is necessary

to point out that all the three classes of functions discussed in this section as well as their self-duality(or anti-self-duality) points and symmetric points can be easily determined by means of Walsh transform.[12,14]

Generalization to the p-valued Logical Functions

The linear transform matrix over GF(p) can be defined as follows:

$$G_p = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1^0 & 1^1 & 1^2 & \ldots & 1^{p-1} \\ 2^0 & 2^1 & 2^2 & \ldots & 2^{p-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ (p-1)^0 & (p-1)^1 & (p-1)^2 & \ldots & (p-1)^{p-1} \end{bmatrix}$$

and

$$G_1 = G_p; \quad G_n = G_p \times G_{n-1} \qquad n=2,3,\ldots \qquad (12)$$

where "$\times$" means Kronecker product of matrices.

It is easy to verify that the inverse of G_p is

$$G_p^{-1} = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & -1^{p-2} & -2^{p-2} & \ldots & -(p-1)^{p-2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & -1 & -2 & \ldots & -(p-1) \\ -1 & -1 & -1 & \ldots & -1 \end{bmatrix}$$

By the properties of the Kronecker product of matrices, we know that

$$G_n^{-1} = G_p^{-1} \times G_{n-1}^{-1} \qquad n = 2,3,\ldots \qquad (13)$$

The linear transform pair can be defined as

$$\begin{cases} F = G_n S \\ S = G_n^{-1} F \end{cases} \qquad (14)$$

If we choose F to be the truth table of the p-valued logical function $f(x)$, the S will be the corresponding generalized Reed-Muller expansion coefficients of $f(x)$. By using the decomposition of the matrix generated by Kronecker product[11], we have the following fast algorithm

Algorithm C:

$$a_0(x) = f(x)$$
$$a_i(t+sp^{n-1}) = \sum_{k=0}^{p-1} g_{sk}a_{i-1}(tp+k) \pmod{p} \qquad (15)$$
$$s(w) = a_n(w)$$

where
$$w,x = 0,1,\ldots,p-1; \quad i = 1,2,\ldots,n;$$
$$s = 0,1,\ldots,p-1; \quad t = 0,1,\ldots,p^{n-1}-1.$$

The g_{sk} in Eq.(15) is the s'th row and k'th column element of G_p or G_p^{-1} respectively corresponding to the forward or inverse transform.

In order that the translation of the arguments be converted into the processing of the spectrum, a translation transform can be defined. Since the

translation of the arguments is equivalent to the permutation of F, let n = 1 and the permutation matrix be P, then

$$S_p = G_1^{-1} PF.$$

Suppose S_p could be obtained from S by a linear transform, i.e.

$$S_p = \Psi S = \Psi G_1^{-1} F = G_1^{-1} PF \tag{16}$$

If Eq.(16) holds for any F, we have

$$\Psi G_1^{-1} = G_1^{-1} P$$

therefore

$$\Psi = G_1^{-1} PG_1 \tag{17}$$

When $n > 1$, it can be proved that the transform matrix corresponding to the translation of the k'th argument is

$$Q_k = I_{p^k} \times Q \times I_{p^{n-k-1}} \tag{18}$$

where Q is the transform matrix for $n = 1$; I_{p^k} is an identity matrix of order p^k; "X" is Kronecker product.

It can be proved that even the orders of the three matrices in Eq.(18) are different, the matrix Ψ_k can still be decomposed. The algorithm based on this decomposition is as follows:

Algorithm D:

1) If $k \neq n-1$, then $a_1(t+sp^{k+1})=s(tp^{n-k-1}+s)$
   $t=0,1,\ldots,p^{k+1}-1; \ s=0,1,\ldots,p^{n-k-1}-1$
   If $k = n-1$, then $a_1(t) = s(t)$
   $t = 0,1,\ldots,p^n-1$

2) $a_2(t+sp^{n-1})=\sum_{k=0}^{p-1} \Psi_{sk} a_1(tp+k) \pmod{p} \tag{19}$
   $t=0,1,\ldots,p^{n-1}-1; \ s=0,1,\ldots,p-1$

3) If $k \neq 0$, then $s_p(t+sp^{n-k}) = a_2(tp^k+s)$
   $t=0,1,\ldots,p^{n-k}-1; \ s=0,1,\ldots,p^k-1$
   If $k=0$, then $s_p(t) = a_2(t)$
   $t = 0,1,\ldots,p^n-1$

It can be proved that if the translation is add 1 (mod p) to the argument(all translation can be realized in this way), then the Q in Eq.(17) will be an up triangular matrix. At this time, the summation in the second step of Eq.(19) can begin from s instead of 0.

Like the case in $p = 2$, some properties of specific class of logical functions can be utilized to reduce the amount of operation needed to find the optimal GMC.

quasi-self-dual functions:
For the given $f(x)$, if there exist a point a and a constant p_0 such that

$$f(x \oplus a) = p_0 \oplus f(x) \pmod{p} \tag{20}$$

then $f(x)$ is said to be a quasi-self-dual function with a quasi-self-duality point a.

It is easy to verify that for $p > 2$ the following relation still exists

$$1 \longrightarrow \delta(w)$$

Therefore, for the quasi-self-duality point p of a quasi-self-dual function $f(x)$ we have

$$S_p(w) = p_0 \delta(w) \oplus s(w)$$

in addition, we can prove that the set of all quasi-self-duality points is an Abelian group(with respect to p-ary addition), and the order of this group is a power of p. In the same way as in $p=2$, this offers some convenience for finding the optimal GMC. The details are somewhat the same as those of $p=2$. The identification of the quasi-self-dual function and the determination of its quasi-self-points can be achieved by means of Chrestenson transform.

Conclusion

In this paper we give out some procedures for finding the optimal GMC expansions for p-valued logical functions($p \geq 2$). The spectrum of a logical function under our transform is just its GMC expansion coefficients. The optimal GMC is obtained by processing these spectra. For $p=2$, the amount of operation for optimization is $(2^n-1)2^{n-1}$ mod 2 additions for a function of n arguments, and this can be reduced a great deal for some special classes of functions. At the same time, the methods here avoid completely the manipulation of character, and the Boolean diffrence of a function. So that they can be easily realized by using of a computer. The limitation that p is a prime is only out of the convenience of the computation. Theoretically it is valid for any finite field. All procedures here have been tested using an Apple II micro-computer. Some examples are given in Appendix.

References

1. A. Mukhopadhyay, "Complete sets of logic primatives," in Recent Development in Switching Theory. New York: Academic Press, 1971.
2. S.M. Reddy,"Easily testable realization for logic functions,"IEEE Trans., vol. C-21, pp.1183-88, Nov. 1972.
3. W.P. Edward,"Minimally testable Reed-Muller Canonical forms," IEEE Trans., vol. C-29, pp.746-750, Aug. 1980.
4. A. Mukhopadhyay and G. Schmitz,"Minimization of Exclusive-OR and logical equivalence switching circuits," IEEE Trans., vol. C-19, pp.132-140, Feb. 1970.
5. M. Davio,"Ring sum expansion of Boolean functions," in Proc. Symp. on Computer and Automata , Polytechnic Inst. of Brooklyn Symp. Proc., vol.

21, pp.411-418, Apr. 1971.

6. G. Bioul, M. Davio and J.P. Deschamps,"Minimization of ring-sum expansion of Boolean functions," Philips Res. Rep., vol. 28, pp.17-36, 1973.

7. A.Thayse,"Integer expansions of discrete functions and their use in optimization problem," Proc. 9'th Int. Symp. Multiple-valued Logic, pp.82-87, 1979.

8. S.B. Marincovic and Z. Tosic,"Algorithms for minimal polarized form determination," IEEE Trans., vol. C-23, pp.1313-1315, Dec. 1974.

9. V.H. Tokmen and S.L. Hurst,"A consideration of universal-modules for ternary synthesis based upon Reed-Muller coefficients," Proc. 9'th Int. Symp. Multiple-valued Logic, pp.248-56, 1979.

10. X. Wu, X. Chen and S.L.Hurst,"Mapping of Reed-Muller coefficients and the minimization of exclusive Or-switching functions,"IEE Proc., vol. 129, Pt.E, No.1, pp.15-20, Jan. 1982.

11. H.C. Andrews and K.L. Caspari,"A generalized Technique for spectral analysis," IEEE Trans., vol. C-19, Jan. 1970.

12. M.G. Karpovsky, Finite Orthogonal Series in the Design of Digital Devices. New York: John Wiley & Sons, 1976.

13. K.L. Kodandapani and R.V. Setlur,"A note on minimal Reed-Muller canonical forms of switching functions," IEEE Trans., Vol.C-26, pp.310-313, Mar. 1977.

14. S.L. Hurst,The Logical Processing of Digital Signals. New York: Crane, Russak; London: Edward Arnold, 1978.

Appendix

I. For a Boolean function $f(x)$, let A denote the set of all self-duality points, B the set of all anti-self-duality points and $V = A \oplus B$.

For any v_1, $v_2 \in V$:

1) If $v_1, v_2 \in A$, then for any $x \in \{0,1,\ldots,2^n-1\}$

$$f(x \oplus v_1 \oplus v_2) = \overline{f(x \oplus v_1)} = f(x)$$

i.e. $v_1 \oplus v_2 \in B$, so that $v_1 \oplus v_2 \in V$;

2) If $v_1 \in A$, $v_2 \in B$, then for any $x \in \{0,1,\ldots,2^n-1\}$

$$f(x \oplus v_1 \oplus v_2) = f(x \oplus v_1) = \overline{f(x)}$$

i.e. $v_1 \oplus v_2 \in A$, so that $v_1 \oplus v_2 \in V$;

3) If $v_1, v_2 \in B$, then for any $x \in \{0,1,\ldots,2^n-1\}$

$$f(x \oplus v_1 \oplus v_2) = f(x \oplus v_1) = f(x)$$

i.e. $v_1 \oplus v_2 \in B$, so that $v_1 \oplus v_2 \in V$.

Summarize all the three points above, it can be seen that V is closed for the operation $\oplus$. Obviously V is a subset of $X = \{0,1,\ldots,2^n-1\}$, and the latter is a finite group for $\oplus$. So V is a subgroup of X, and its order is a power of 2.

II. Examples of optimization.

Example 1:
Let $n = 3$, $p = 2$ and $F = (10001110)$. By Eq.(5) we find that $S = (11110110)$. The procedure for finding the optimal coefficients and optimal polarity is as follows:

Polarities(Gray code) S_p

Polarities	S_p
0 0 0	1 1 1 1 0 1 1 0
0 0 1	0 1 0 1 1 1 1 0
0 1 1	0 0 0 1 0 1 1 0
0 1 0	0 0 1 1 1 1 1 0
1 1 0	1 1 0 1 1 1 1 0
1 1 1	0 1 1 1 0 1 1 0
1 0 1	1 0 1 1 1 1 1 0
1 0 0	1 0 0 1 0 1 1 0

The underlined sections are to be added to the corresponding sections without underline. We can see that $p_0 = 3(011)$ gives minimum weight of S_p. By Eq.(6') we have

$$f(x \oplus 3) = x_1 x_2 + x_0 x_1 + x_0 x_2 \qquad (\text{mod } 2)$$

therefore we have its optimal RMC expansion

$$f(x) = \overline{x}_1 \overline{x}_2 + x_0 \overline{x}_1 + x_0 \overline{x}_2. \qquad (\text{mod } 2)$$

Example 2:
Let $n = 2$, $p = 3$ and $F = (010111000)$. By Eq.(15) we have $S = (022200211)$. Using algorithm D we can find that the optimal translation point is $p_0 = 7$ (21), and $S_7 = (000001101)$. Therefore we have

$$f(x \oplus 7) = x_0 x_1^2 + x_0^2 + x_0^2 x_1^2 \qquad (\text{mod } 3)$$

and

$$f(x) = (x_0-2)(x_1-1)^2 + (x_0-2)^2 + (x_0-2)^2(x_1-1)^2$$

$$(\text{mod } 3).$$

Session 8A
Detection and Diagnosis

PREVIOUS PAGE
IS BLANK

RELATIONS AMONG SYSTEM DIAGNOSIS

MODELS WITH THREE-VALUED TEST OUTCOMES

Jon T. Butler*

Department of Electrical Engineering
and Computer Science
Northwestern University
Evanston, IL 60201

ABSTRACT

Three models of multiprocessing systems are compared on the basis of the accuracy of the diagnosis of faulty processors. One model is the conventional system with binary-valued test outcomes (pass and fail). The other models have three-valued test outcomes, where the third value is either a missing or an incorrect test result. It is shown that, in general multiprocessing systems, there is a hierarchy among the three models with respect to diagnosability. However, in systems where no two processors test each other, the models are on par, and established criteria for diagnosability in binary systems can be used in both of the three-valued systems.

I. INTRODUCTION.

Fault tolerance in a multiprocessing system can be achieved by allowing processors to test each other. Such a system can be modeled as a directed graph, where nodes represent processors and arcs represent tests. This representation, introduced by Preparata, Metze, and Chien [1] in 1967, has received considerable attention. Hakimi and Amin [2] derived a complete characterizaton of the tests (arcs) needed to guarantee the unique identification of a minimum number of faulty nodes. Mallela and Masson [3,4] extended the model to include intermittent, as well as permanent failures. Other studies have focused on an alternative form of test invalidation [5], various diagnosis strategies [6,7,8,9,10], and a model in which test outcomes are generated from a comparison of job results [11].

In most studies, it is assumed that the test outcomes are binary-valued, with 0 and 1 representing pass and fail, respectively. Recently, the model has been extended to accommodate test outcomes which are not available due to incomplete testing or to faulty transmission [12]. That is,

test results take on a third value, 2, corresponding to a missing test result.

In this paper, a three-valued system is considered in which 0 and 1 represent pass and fail, as before, while the third value, 0', denotes an incorrect pass outcome. Such outcomes occur when a fault-free processor tests an intermittently faulty processor that happens to be fault-free at the time the test is applied. As with test results produced by faulty processors and missing test results, 0' results complicate the diagnosis. In fact, the presence of such results is especially troublesome. It is shown, for example, that systems exist in which an accurate diagnosis can proceed if some number, τ, of the test results are missing, while the presence of τ incorrect results, on the other hand, does <u>not</u> allow an accurate diagnosis.

This paper focuses on the relationship between the two three-valued systems. A set of systems which can tolerate the same number of incorrect as missing test results is shown. Further, a relationship between the two three-valued systems and conventional binary systems is developed.

II. BACKGROUND.

A **system** is a directed graph where the set of nodes $\{u_0, u_1, \ldots, u_n\}$ represents **processors** and the arcs represent **tests** between processors. In particular, u_i tests u_j if there exists a directed arc from u_i to u_j. The **test outcome** is **0** or **1** if the test result is **pass** or **fail**, respectively, and is **2** if the test outcome is **missing**.

If a processor is **permanently faulty**, it will fail all tests of it by fault-free processors and will produce either a 0 or 1 result for all tests it applies to other processors. An **intermittently faulty** processor u_i may, on the other hand, pass a test by a fault-free processor, since the test may have been applied when u_i was fault-free. Such a test result will be denoted as an **incorrect pass** and represented as **0'**. It should be noted that during diagnosis, a 0' is seen as a 0. Further, an intermittently faulty processor will produce

either a 0 or 1 test result for all processors it tests.

As an example, consider Fig. 1, which shows a three-processor system for two cases,

1. a single permanently faulty processor (Fig. 1a) and
2. a single intermittently faulty processor (Fig. 1b).
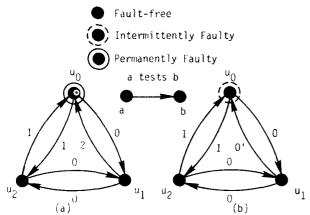
For each case, a possible a set of test outcomes

● Fault-free

◉ Intermittently Faulty

◎ Permanently Faulty

Figure 1. A System S_1 Which is $1/_2 1$-Diagnosable But Not $1/_0 1$-Diagnosable.

is shown. In both examples, the faulty processor fails a test by one fault-free processor and passes the other.

A system is $t'/_2\tau$-**diagnosable** if all faulty processors can be uniquely identified provided there are t' or fewer faulty processors and τ or fewer missing test results. For example, the system of Fig. 1 is $1/_2 1$-diagnosable as follows. If there were no missing results, 1. the single faulty processor would produce fail test outcomes for the two tests applied to it and 2. the two fault-free processors would pass the tests applied to each other. No other pair of processors will produce a pair of pass test outcomes. Since the four tests covered by these two conditions are distinct, even if one result is missing, there is still enough information to specify the faulty processor uniquely.

A system is $t'/_0\tau$-**diagnosable** if all faulty processors can be uniquely identified providing there are t' or fewer faulty processors and τ or fewer incorrect test results. Note that the system of Fig. 1 is **not** $1/_0 1$-diagnosable as follows. Fig. 1b shows that u_0 is intermittently faulty, and the test by u_1 is an incorrect pass, $0'$. However, in the diagnosis, neither the faulty condition of u_0 nor the prime on the pass test of u_0 by u_1 is seen. If we assume at most one processor is faulty, we will be unable to determine whether it is u_0 or u_2.

When $\tau = 0$, $t'/_2$- and $t'/_0\tau$-diagnosability are identical. This special case will be referred to as **t-diagnosability**.

III. RELATIONSHIP BETWEEN DIAGNOSABILITY OF GENERAL SYSTEMS.

In this section, we are interested in determining the relation between the various measures of system diagnosis. For example, we are interested in how a system's tolerance to incorrect test results determines its tolerance to missing results. From the previous section, we can write,

Observation 1: There exists a system (S_1) which is $t'/_2\tau$-diagnosable but not $t'/_0\tau$-diagnosable, for $t',\tau \geq 1$.

This result indicates that incorrect information is more debilitating than the same amount of missing information. In effect, incorrect pass test results introduce a degree of ambiguity not realizable by the same number of missing test results.

From the converse point of view, we have

Theorem 1: If S is $t'/_0\tau$-diagnosable, then S is $t'/_2\tau$-diagnosable, for $t',\tau \geq 1$.

Proof: See Appendix I.

Thus, from Theorem 1, it follows that if t' intermittently and permanently faulty processors can be uniquely identified in the presence of τ or fewer incorrect pass test results, then t' or fewer permanently faulty processors can be uniquely identified in the presence of τ or fewer missing test results.

Consider the relation between $t'/_2\tau$-, $t'/_0\tau$-, and t-diagnosability. From [12], we have the following.

Observation 2 [12]: There exists a system (S_1) which is $t'/_2\tau$-diagnosable but not $(t'+\tau)$-diagnosable, for some $t', \tau \geq 1$.

This observation follows from the fact that any arrangement of two permanently faulty processors in system S_1 of Fig. 1 can produce the set of test outcomes in which all test results are 1. Because of symmetry, it is impossible to determine the faulty processors from the set of test outcomes. However, from [12] the converse relation holds,

Theorem 2 [12]: If S is t-diagnosable, then S is $t'/_2\tau$-diagnosable, where $t' + \tau = t$.

In the hierarchy of the types of diagnosability, $t'/_2\tau$-diagnosability represents the "weakest" form. That is, from Theorem 1, $t'/_0\tau$-diagnosability implies $t'/_2\tau$-diagnosability, while the converse is not true (Observation 1). From the above result, $(t'+\tau)$-diagnosability implies $t'/_2\tau$-diagnosability, while the converse is also not true (Observation 2). It is of interest, therefore, to consider the relationship between t- and $t'/_0\tau$-diagnosability. We have,

Observation 3: There exists a system (S_2) which is t-diagnosable but not $t'/_0\tau$-diagnosable, where $t' + \tau = t$.

351

To see this, consider the system S_2 shown in Fig. 2. S_2 consists of a system $D_4(9)$ plus two additional nodes, u_9 and u_{10}. $D_4(9)$, shown as a
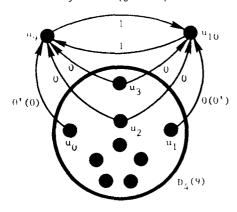

Figure 2. A System S_2 Which is 4-Diagnosable But Not 3/1-Diagnosable.

circle, is known to be 4-diagnosable [1]. It is claimed that S_2 is also 4-diagnosable. With at most 4 faulty processors, there are three cases.

1. Both u_9 and u_{10} are fault-free. Thus, all faulty processors must be in $D_4(9)$. Since there are 4 or fewer faulty processors and $D_4(9)$ is 4-diagnosable, they can be uniquely identified.

2. One of u_9 and u_{10} is faulty. If there are 3 faulty processors in $D_4(9)$, they can be uniquely identified. Further, at least one of u_9 and u_{10} is tested by a known fault-free processor. If it is faulty, we are done, since, by assumption, there are 4 or fewer faulty processors. If it is fault-free, its test of the other processor will determine that the latter is faulty. If there are 2 or fewer faulty processors in $D_4(9)$, then both u_9 and u_{10} are tested by known fault-free processors, and we are done.

3. Both u_9 and u_{10} are faulty. Thus, there can be no more than two faulty processors in $D_4(9)$ and, so both u_9 and u_{10} are tested by known fault-free processors.

We now show that S_2 is not 3/1-diagnosable. Consider the case where both u_2 and u_3 are faulty and one of u_9 and u_{10} is intermittently faulty. Let the set of test outcomes of these two processors be as shown in Fig. 2. If u_9 is the faulty processor, the result of the test by u_0 of u_9 is an incorrect pass (outcomes outside the parantheses); otherwise, the result of the test by u_1 of u_{10} is an incorrect pass (outcomes inside the parantheses). Thus, there are three faulty processors and one incorrect pass. Since it is impossible to determine which of u_9 or u_{10} is faulty, S_2 is not 3/1-diagnosable. This proves Observation 3.

With respect to the converse, we have,

Observation 4: There exists a system (S_3) which is t'/τ-diagnosable but not $(t'+\tau)$-diagnosable, for some $t', \tau \geq 1$.

Fig. 3 shows S_3, a complete digraph on n = 5 nodes. S_3 is not 3-diagnosable, since it violates the condition $n \geq 2t + 1$ shown in [1] to be necessary
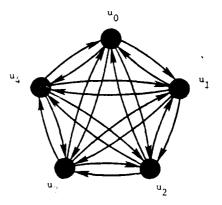

Figure 3. A System S_3 Which is 1/2-Diagnosable But Not 3-Diagnosable.

for an n processor system to be t-diagnosable. However, S_3 is 1/2-diagnosable as follows. If all test outcomes by fault-free processors were correct, then a single faulty processor would fail all four tests applied to it, while all other (fault-free) processors would fail at most one test. If there are at most two test outcomes of tests of the faulty processors which are incorrect passes, then the faulty processor fails at least *two tests and can, thus, be identified*. This confirms Observation 4. Because of the large number of tests per processor, even if several are incorrect passes, sufficiently many valid test results remain to determine which processor is faulty.

In the hierarchy mentioned earlier t'/τ- and $(t'+\tau)$-diagnosability are on par. We can make no general statement about implication of the two properties one way or the other. The results of this section are summarized in Fig. 4. The circles correspond to the three types of diagnosability, while the arrows represent implication.
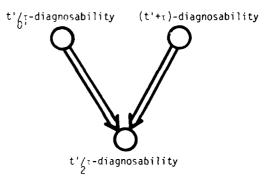

Figure 4. Summary of Results For General Systems.

IV. RELATIONSHIP BETWEEN DIAGNOSABILITY IN SYSTEMS WHERE NO TWO PROCESSORS TEST EACH OTHER.

Hakimi and Amin [2] have shown that systems with permanently faulty processors and no missing test outcomes in which no two processors test each other have a t-diagnosability which is exactly prescribed by two simple conditions. In this section, we show that such systems have a special property with respect to the three types of diagnosability. For example, with respect to $t'/_2\tau$ - and $(t'+\tau)$- diagnosability, we have,

Theorem 3 [12]: Let S be a system in which no two processors test each other. S is $t'/_2\tau$-diagnosable iff S is $(t'+\tau)$-diagnosable.

Recall from Observation 2 that, in general systems, there exists a system which is $t'/_2\tau$-diagnosable, but not $(t'+\tau)$-diagnosable. From the above result, this implies that, in such systems, there is at least one pair of processors which test each other.

Consider now $t'/_6\tau$- and $(t'+\tau)$-diagnosable systems. We have,

Theorem 4: Let S be a system in which no two processors test each other. S is $t'/_6\tau$-diagnosable iff S is $(t'+\tau)$-diagnosable.

Proof: See Appendix II.

Analogous to $t'/_2\tau$-diagnosability, the only examples of systems which are $t'/_6\pi$-diagnosable but not $(t'+\tau)$-diagnosable are found in the set of systems where there exists a pair of processors which test each other.

The relation between $t'/_2\tau$- and $t'/_6\pi$-diagnosability for systems in which no two processors test each other can be seen immediately from Theorems 3 and 4.

Theorem 5: Let S be a system in which no two processors test each other. S is $t'/_2\tau$-diagnosable iff S is $t'/_6\tau$-diagnosable.

Theorem 3 establishes the fact that the set of systems in which no two processors test each other, the subset which is $t'/_2\tau$-diagnosable is exactly the subset which is $t'/_6\tau$-diagnosable. It follows from this result and Observation 2 that, over all systems, those which are $t'/_2\tau$-diagnosable, but not $t'/_6\pi$-diagnosable contain at least one pair of processors which test each other.

The results of this section are summarized in the diagram of Fig. 5. The double headed arrows represents the iff of each theorem. The significance of this result is that a test for one type of diagnosability is a test for the other two types. In particular, since a simple test T for $(t'+\tau)$-diagnosability exists for systems in which no two processors test each other [2], T can also be used to test for $t'/_2\tau$- and $t'/_6\pi$-diagnosability. (T is 1. $n \geq 2t + 1$ and 2. all processors are tested by at least t other processors for $t = t' + \tau$.)
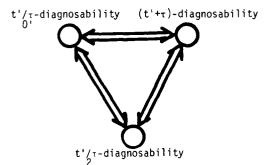

Figure 5. Summary of Results For Systems in Which No Two Processors Test Each Other.

V. CONCLUDING REMARKS.

This paper has considered three types of diagnosability

1. $(t'+\tau)$-diagnosability corresponding to permanently faulty processors with binary test outcomes; 0 (pass) and 1 (fail),

2. $t'/_2\tau$-diagnosability corresponding to permanently faulty processors with three-valued test outcomes; 0 (pass), 1 (pass), and 2 (missing), and

3. $t'/_6\tau$-diagnosability corresponding to permanently and intermittently faulty processors with three-valued test outcomes; 0 (pass), 1 (fail), and 0' (incorrect pass).

It is shown that for a general multiprocessing system S, if S is either $t'/_6\tau$- or $(t'+\tau)$-diagnosable, it is also $t'/_2\tau$-diagnosable. In general, no further statements of this type can be made.

However, in a system S where no two processors test each other, if S possesses one of the three types of diagnosability, it possesses them all. As there are no simple tests for $t'/_2\tau$- and $t'/_6\pi$-diagnosability, tests for $(t'+\tau)$-diagnosability can be used in systems where no two processors test each other.

REFERENCES

[1] F. P. Preparata, G. Metze, and R. T. Chien, "On the connection assignment problem of diagnosable systems," IEEE Transactions on Computers, C-16, pp. 848-859, December 1967.

[2] S. L. Hakimi and A. T. Amin, "Characterization of the connection assignment of diagnosable systems," IEEE Transactions on Computers, C-23, pp. 86-88, January 1974.

[3] S. Mallela and G. M. Masson, "Diagnosable systems for intermittent faults," IEEE Transactions on Computers, C-27, pp. 560-566, June 1976.

[4] S. Mallela and G. M. Masson, "Diagnosis without repair for hybrid fault situations," *IEEE Transactions on Computers*, C-29, pp. 461-468, June 1980.

[5] F. Barsi, F. Grandoni, and P. Maestrini, "A theory of diagnosability of digital systems." *IEEE Transactions on Computers*, C-25, pp. 585-593, June 1976.

[6] J. E. Smith, "Universal systems diagnosis algorithms," *IEEE Transactions on Computers*, C-27, pp. 374-378, May 1979.

[7] J. T. Butler, "Speed-efficiency-complexity tradeoffs in universal diagnosis algorithms," *IEEE Transactions on Computers*, C-30, pp. 590-595, August 1981.

[8] J. R. Armstrong and F. G. Gray, "A multivalued, global, majority voter," *Proc. of the 10th International Symposium on Multiple-Valued Logic*, pp. 268-271, June 1980.

[9] M. Adham and A. D. Friedman, "Digital system fault diagnosis," *Jour. Design Automation and Fault Tolerant Computing*, vol. 1, pp. 115-132, Feb. 1977.

[10] J. T. Butler, "Diagnosis of intermittently faulty and permanently faulty processors in a multiprocessing system using three-valued functions," *Proc. of the 12th International Symposium on Multiple-Valued Logic*, pp. 122-128, May 1982.

[11] K.-Y. Chwa and S. L. Hakimi, "Schemes for fault-tolerant computing: a comparison of modularly redundant and t-diagnosable systems," *Information and Control*, 49, pp. 212-238, June 1981.

[12] J. T. Butler, "Properties of three-valued systems diagnosis," *Proc. of the 11th International Symposium on Multiple-Valued Logic*, pp. 85-89, May 1981.

APPENDIX I - PROOF OF THEOREM 1

Theorem 1: If S is $t'_6\tau$-diagnosable, then S is $t'_2\tau$-diagnosable, for $t', \geq 1$.

Proof: On the contrary, assume there is a system S which is $t'_6\tau$- but not $t'_2\tau$-diagnosable. Since S is not $t'_2\tau$-diagnosable, there are two fault patterns, FP_1 and FP_2, each with t' or fewer faulty processors that produce the same set q_1 of test outcomes in which there are τ or fewer 2's. We show that there is a set q_2 of test outcomes with τ or fewer 0' outcomes of tests of faulty processors by fault-free processors produced by FP_1 and FP_2. Thus, S is not $t'_6\tau$-diagnosable, as assumed.

q_2 is identical to q_1 except that all 2's are replaced by 0's or 0''s. Although some of these tests may be by fault-free processors on faulty processors, there will be no more than

of them. Thus, two fault patterns, FP_1 and FP_1', each with τ or fewer faulty processors, produces the same set of test outcomes containing no more than τ tests of faulty processors by fault-free processors which produce a 0 test result. Thus, S is not $t'_6\tau$-diagnosable.

Q.E.D.

APPENDIX II - PROOF OF THEOREM 4

Theorem 4: Let S be a system in which no two processors test each other. S is $t'_6\tau$-diagnosable iff S is $(t'+\tau)$-diagnosable.

Proof: (if) Let S be a t-diagnosable system in which no two processors test each other. S is shown to be $t'_6\tau$-diagnosable for $t = t'+\tau$ by contradiction. That is, assume, on the contrary, S is not $t'_6\tau$-diagnosable. We proceed by showing that not all processors in S are tested by $t'+\tau$ other processors, and so from [1], S is not $(t'+\tau)$-diagnosable, as assumed.

If S is not $t'_6\tau$-diagnosable, then there exists two fault patterns, FP_1 and FP_2, each with t' or fewer faulty processors, which produce the same set of test outcomes containing no more than τ incorrect pass test outcomes. Fig. 6 shows the situation.
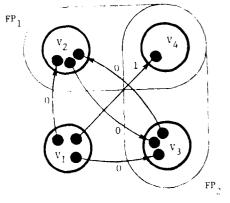

Figure 6. A System Which is $(t'+\tau)$-Diagnosable But Not $t'_6\tau$-Diagnosable.

Here $FP_1 = V_2 \cup V_4$ and $FP_2 = V_3 \cup V_4$. Let $\Gamma_{in}(V)$ be the number of tests applied to processors in V. An upper bound on $\Gamma_{in}(V_2 \cup V_3)$ can be expressed as

$$\Gamma_{in}(V_2 \cup V_3) \leq \tau_1 + \tau_2 + |V_2 \cup V_3| |V_4| + \tag{1}$$
$$|V_2 + V_3|(|V_2 \cup V_3|-1)/2,$$

where $\tau_1(\leq\tau)$ and $\tau_2(\leq\tau)$ are the number of incorrect pass outcomes of tests applied to V_2 and V_3, respectively. $\tau_1 + \tau_2$ represents the maximum number of tests which can be applied by processors in V_1. $|V_2 \cup V_3| |V_4|$ represents the maximum number of tests which can be applied by

processors in V_4. $|V_2 \cup V_3|(|V_2 \cup V_3|-1)/2$ represents the maximum number of tests applied by processors in $V_2 \cup V_3$, there being no two processors which test each other. Consider two cases.

Case 1: $|V_2 \cup V_3| \geq 2$. Dividing both sides of (1) by $|V_2 \cup V_3|$ yields

$$\frac{\Gamma_{in}(V_2 \cup V_3)}{|V_2 \cup V_3|} \leq \frac{\tau_1 + \tau_2}{|V_2 \cup V_3|} + \frac{2|V_4| + |V_2 \cup V_3| - 1}{2}.$$

Since $|V_2 \cup V_3| \geq 2$ and $|V_2 \cup V_3| \leq |V_2| + |V_3|$, we have,

$$\frac{\Gamma_{in}(V_2 \cup V_3)}{|V_2 \cup V_3|} \leq \frac{(\tau_1 + |V_3| + |V_4|) + (\tau_2 + |V_2| + |V_4|) - 1}{2}$$

$$\leq (t' + \tau) - 1/2. \qquad (2)$$

Because the average number of tests per processor in $V_2 \cup V_3$, as expressed on the left side of (2) is less than $t' + \tau$, at least one is tested by fewer than $t' + \tau$ processors. Thus, S is not $(t'+\tau)$-diagnosable.

Case 2: $|V_2 \cup V_3| = 1$. Assume that $|V_2| = 1$ and $|V_3| = 0$. We have,

$$L(V_2) < \tau_1 + |V_4|$$

and there are two fault patterns, one with $|V_4|$ faulty processors and one with $|V_4| + 1$ faulty processors which produce the same set of test outcomes. Thus, S is not $(t' + \tau)$-diagnosable, where $t' + \tau = |V_4| + 1 + \tau_1$.

(only if) One can show easily that in a $t' + \tau$-diagnosable system each processor is tested by at least $t' + \tau$ other processors. This completes the proof, since such a system is $(t'+\tau)$-diagnosable when no two processors test each other [2].

$$Q.E.D.$$

355

ON SYSTEM DIAGNOSIS WITH MULTIVALUED TEST OUTCOMES

A. Sen Gupta
Quantitative & Information Science Department
Western Illinois University
Macomb, Illinois 61455

and

A. Sen
Computer Science Unit
Indian Statistical Institute
Calcutta, India

Abstract

The problem of diagnosis of systems when the test outcomes are multivalued was considered earlier[1]. However, a too strong sufficient condition was specified there for the diagnosability of such a system, which was shown not to be a necessary one. Necessary and sufficient conditions for the diagnosability of a system under multivalued test outcomes have been presented in this paper which is applicable to any arbitrary system.

Introduction

Considerable research has been reported in the literature about the diagnosability of a self-diagnosable system. However, most of these relate to testing the diagnosability of a system when the test outcomes are binary. Butler[1] presented some properties of system in connection to its diagnosability in case of three valued test outcomes. *Under the assumption that no two components of the given system test each other, necessary and sufficient condition for the diagnosability of the system was derived[1].* A too strong sufficient condition for diagnosability of a system was derived there[1] which was shown to be not a necessary one.

In this paper, we present a necessary and sufficient condition to be satisfied by any arbitrary self-diagnosable system in order that it be diagnosable in presence of three-valued test outcomes. We assume the system to be represented by the PMC model[2] and each of the test outcomes is either of the three values 0, 1, and 2. The outcome 0 represents the tested component to pass the test from the testing component, the outcome 1 represents it fails and 2 represents the information of pass or fail is missing. Thus, in case, when all the faults of the components of the system are of permanent nature, the outcome 1 means if the testing component is fault-free, the tested component is faulty, 0 means if the testing component is fault-free, the tested component is also fault-free; but no inference can be drawn about the tested component if the testing component itself is faulty and in this case, the outcome may be either of 0 or 1, irrespective of the nature of the tested component. The test outcome 2 can occur for any testing process irrespective of the nature of the testing and tested components, when this information for pass or fail is somehow not available. Henceforth we will refer a complete set of test outcome

as a syndrome.

The problem of diagnosability of a system arises because of the fact that the same syndrome may be produced by two distinct fault situations, i.e., for two distinct sets of faulty components. A system is said to be t/x - diagnosable, if from the syndrome, the faulty components can be uniquely identified, provided not more than t components are faulty and not more than x test outcomes are missing. It has been shown[1] that if a system is (t+x) - diagnosable as defined in PMC model[2], it is t/x - diagnosable. However, this is too strong a sufficient condition and is not a necessary condition also.

However, if some of the components of the system may fail intermittently, a faulty component may pass test from a fault-free testing component, possibly because the faulty component is intermittently faulty and was working in a fault-free manner when the testing was carried out. A three valued model for such a situation has been presented[3], where the test outcomes are 0, 1, and 0^1 where 0 and 1 outcomes are produced under the same condition as in a permanent fault situation and 0^1 is the test outcome when an intermittently failing component passes a test from a fault-free component. For diagnosis purpose, of course, 0^1 behaves just like a 0. In this three-valued model, a system is defined to be t[x] - diagnosable if the faulty components can be uniquely identified from a given syndrome provided not more than t components are faulty and there are no more than x outcomes each of which is 0^1. In other words, a system is t[x] - diagnosable if no pair of sets of faulty components produce syndromes which are identical when each 0^1 is replaced by a 0. Testing the t[x] - diagnosability of a system, presented earlier[3], concerns only with the situation when no two components of the system test each other.

In this paper, we present methods for testing the t/x - diagnosability and t[x] - diagnosability of any arbitrary system.

Testing for t/x - diagnosability

Let S represent the set of components of a system. For any $S_1 \in S$, if the fault situation is such that all the components in S_1 are faulty and those in $S - S_1$ are fault-free, usually a number

of syndromes may exist corresponding to this fault situation. If for two subsets S_1, S_2 of S, there exists a syndrome which is a possible syndrome in the two fault situations given by the sets of faulty components as S_1 and S_2 respectively, then the two sets S_1 and S_2 are referred as indistinguishable faulty sets. Thus, in order that a system may be t/x - diagnosable, i.e., the faulty components can be uniquely identified from the syndrome, provided not more than t components are faulty, we must have for every pair of sets S_1, $S_2 \subseteq S$, such that, $|S_1|$, $|S_2| \leq t$, S_1 and S_2 must be distinguishable faulty sets.

For any two subsets S_1, S_2 of S, we define a function as $T(S_1, S_2)$ as follows. In any syndrome, $T(S_1, S_2)$ gives the number of test outcomes, in each of which the tested component belongs to S_2 and the testing component belongs to S_1. Obviously, for a given S_1, S_2, $T(S_1, S_2)$ depends only on the testing feature of the components of the system and not on any syndrome.

<u>Lemma 1</u>: If no more than x outcomes are missing, then two sets of components S_1 and S_2 are distinguishable faulty sets if and only if $T(S - S_1 - S_2$, $(S_1 - S_2) \cup (S_2 - S_1)) > x$.

<u>Proof</u>: <u>Necessity</u>: Let us assume that the condition is not satisfied. Let $T(S - S_1 - S_2$, $(S_1 - S_2) \cup (S_2 - S_1)) = m \leq x$. Consider a syndrome as follows.
(a) An outcome 2 for each component of $(S_1 - S_2) \cup (S_2 - S_1)$ tested by components from $S - S_1 - S_2$.
(b) An outcome 1 for each component of S_2 tested by components of $S_1 - S_2$ and for each component of S_1 tested by components from $S_2 - S_1$ and for each component of $S_1 \cap S_2$ tested by components from $S - S_1 - S_2$.
(c) An outcome 0 for each component of $S - S_2$ tested by components from $S - S_2$ and for each component of $S - S_1$ tested by components from $S - S_1$, except for the outcomes given by (a) above.
(d) The remaining outcomes are arbitrary.

For such a syndrome, no more than x outcomes are having missing values and it may be observed that such a syndrome may be produced when the set of faulty components is either S_1 or S_2. Thus, S_1 and S_2 are indistinguishable faulty sets.

<u>Sufficiency</u>: If $T(S - S_1 - S_2$, $(S_1 - S_2) \cup (S_2 - S_1)) > x$, then all the test outcomes in each of

which the tested components belongs to $(S_1 - S_2) \cup (S_2 - S_1)$ and the testing component belongs to $S - S_1 - S_2$ cannot have missing values. Let $\mu \in S - S_1 - S_2$ and $\nu \in (S_1 - S_2) \cup (S_2 - S_1)$ be such that the test outcome when μ tests ν is not 2. If it is 0, then if $\nu \in S_1 - S_2$, then S_1 cannot be the faulty set of components, otherwise, if $\nu \in S_2 - S_1$, then S_2 cannot be the faulty set of components. If it is 1, then if $\nu \in S_1 - S_2$, then S_1 must be the faulty set of components, otherwise, if $\nu \in S_2 - S_1$, then S_2 must be the faulty set of components. Thus, from this test outcome, we can identify between S_1 and S_2 which is the faulty set of components, i.e., S_1 and S_2 are distinguishable.

From Lemma 1, we can find out the necessary and sufficient conditions for the diagnosability of a system. Since, none of the faulty sets can contain more than t components and the system is t/x - diagnosable when every pair of sets of possible faulty components is distinguishable, we have the following theorem.

<u>Theorem 1</u>: A system is t/x - diagnosable, if and only if, for every S_1, $S_2 \subseteq S$, where S is the set of components of the system, $T(S - S_1 - S_2$, $(S_1 - S_2) \cup (S_2 - S_1)) > x$, whenever $|S_1|$, $|S_2| \leq t$, where $|X|$ represents cardinality of the set X.

However, given any arbitrary system, in order to test for its t/x - diagnosability, we need not have to consider every pair S_1, S_2, such that $|S_1|$, $|S_2| \leq t$ and S_1, $S_2 \subseteq S$. We shall show that consideration of every pair S_1, S_2, such that $|S_1| = |S_2| = t$, will be sufficient. Before we prove this result, we present the following lemma (which gives a necessary condition for t/x - diagnosability) which we will need in our proof.

<u>Lemma 2</u>: If a system is t/x - diagnosable, then every component of the system must be tested by at least $t+x$ other components.

<u>Proof</u>: Let us assume on the contrary. Let ν be a component and let S_0 be the set of components testing ν, such that $|S_0| < t + x$. Form S_a, S_b, such that $S_a \cup S_b = S_0$, $S_a \cap S_b = \emptyset$ and $|S_b| \leq x$ and $|S_a| < t$. Then let $S_1 = S_a \cup \{\nu\}$ and $S_2 = S_a$. Then, $(S_1 - S_2) \cup (S_2 - S_1) = \{\nu\}$, and $T(S - S_1 - S_2$, $(S_1 - S_2) \cup (S_2 - S_1)) \leq x$ and $|S_1|$, $|S_2| \leq t$ and hence, by Theorem 1, the system is not t/x - diagnosable.

<u>Lemma 3</u>: If for every S_1, S_2, such that $|S_1| = t$ and S_1, $S_2 \subseteq S$, $T(S - S_1 - S_2$, $(S_1 - S_2) \cup$

$(S_2 - S_1)) > x$, then, for every S_a, S_b, such that $|S_a|$, $|S_b| \leq t$ and S_a, $S_b \subset S$, $T(S - S_a - S_b, (S_a - S_b) \cup (S_b - S_a)) > x$.

Proof: We will have to consider three cases.

Case I: $S_a \subset S_b$. Consider any $v \in S_b - S_a$. Since v is tested by at least $t+x$ components of which maximum $t-1$ components may belong to S_b, there are at least $x+1$ components in $S - S_b$ which test v. Now, because $S_a \subset S_b$, $S - S_a - S_b = S - S_b$ and $(S_a - S_b) \cup (S_b - S_a) = S_b - S_a$ and $v \in S_b - S_a$, $T(S - S_a - S_b, (S_a - S_b) \cup (S_b - S_a)) \geq x + 1$. Hence Theorem 1 is satisfied for all S_a, S_b whenever $S_a \subset S_b$, $|S_b| \leq t$ and Lemma 2 is satisfied.

Case II. $S_a \not\subset S_b$ and $|S_a| < |S_b|$. Let $S_c \subset S_b - S_a$ such that $|S_c \cup S_a| = |S_b|$. Form $S_a^1 = S_c \cup S_a$. Let $S_d \subset S - S_a^1 - S_b$, such that $|S_d| + |S_b| = t$. Form $S_{a1} = S_a^1 \cup S_d$ and $S_{b1} = S_b \cup S_d$. Then each of S_{a1} and S_{b1} has t elements and $(S_a - S_b) \cup (S_b - S_a) \supset (S_{a1} - S_{b1}) \cup (S_{b1} - S_{a1})$ and $S - S_a - S_b \supset S - S_{a1} - S_{b1}$. But $T(S - S_{a1} - S_{b1}, (S_{a1} - S_{b1}) \cup (S_{b1} - S_{a1})) > x$ as $|S_{a1}| = |S_{b1}| = t$, hence $T(S - S_a - S_b, (S_a - S_b) \cup (S_b - S_a)) > x$ holds good.

Case III. $|S_a| = |S_b|$. Using the same argument as in Case II and using S_a instead of S_a^1, we will have $T(S - S_a - S_b, (S_a - S_b) \cup (S_b - S_a)) > x$.

According to Lemma 3, we should check whether the condition given by Theorem 1 is satisfied only for all S_1, S_2 such that $|S_1| = |S_2| = t$, in order that the system is t/x - diagnosable. Combining Lemma 3 and Theorem 1 we can specify the necessary and sufficient condition for a system to be t/x - diagnosable as follows.

Theorem 2: A system is t/x - diagnosable if and only if, for all S_1, S_2, such that $|S_1| = |S_2| = t$, and S_1, $S_2 \subset S$

$$T(S - S_1 - S_2, (S_1 - S_2) \cup (S_2 - S_1)) > x.$$

Testing for t[x] - diagnosability

If some of the components of a system may fail intermittently, the diagnosis of the faulty components becomes quite complicated because an intermittently failing component may pass a test from a fault-free component. The diagnosis of faulty components from the syndrome in presence of intermittent failures was studied[4-6] assuming an upper bound on the number of intermittently faulty components and all syndromes to be compatible with permanent fault situation (i.e., can be produced by some permanent fault situation). The identification of the faulty components from the syndrome when the number of 0^1 outcomes does not exceed some quantity x and the number of faulty components does not exceed some quantity t will be discussed in this section. For diagnosis purpose, however, 0^1 behaves exactly as 0, i.e., when the faulty components are identified from the syndrome the outcomes are either pass (0) or fail (1).

In order that the faulty components can be uniquely identified from the syndrome, when some of the faulty components may fail intermittently, i.e., some of the test outcomes may be 0^1, there should not exist two distinct subsets S_1 and S_2 of S such that when S_1 and S_2 are the sets of faulty components, there exist two syndromes produced by them respectively which are identical when in each syndrome each 0^1 is replaced by a 0 and the number of 0^1 in each syndrome does not exceed x. We will refer a pair of sets (S_1, S_2), such that S_1, $S_2 \subset S$ and $|S_1|$, $|S_2| \leq t$, as an indistinguishable pair, if each produces a syndrome, which are identical as each 0^1 is replaced by a 0. Hence, a system is t[x] - diagnosable if every pair (S_1, S_2) as above is a distinguishable pair with no more than x outcomes as 0^1 in each of them.

Lemma 4: (S_1, S_2) is a distinguishable pair, if and only if, either of the following conditions is satisfied.

   (i) $T(S - S_1 - S_2, S_1 - S_2) > x$
   (ii) $T(S - S_1 - S_2, S_2 - S_1) > x$

Proof: Necessity: Suppose neither of the two conditions is satisfied. Let S_a be the smallest subset of such that $T(S - S_1 - S_2, S_1 - S_2) = T(S - S_1 - S_2, S_a)$, i.e., S_a contains all those components of $S_1 - S_2$ which are tested by components from $S - S_1 - S_2$. Similarly, let S_b be the smallest subset of $S_2 - S_1$, such that $T(S - S_1 - S_2, S_2 - S_1) = T(S - S_1 - S_2, S_b)$. Let $S_{ax} = S_1 - S_a$ and $S_{bx} = S_2 - S_b$. Consider a syndrome as follows:
(a) An outcome 0^1 for each component of S_a tested by components from $S - S_1 - S_2$.
(b) An outcome 1 for each component of S_{ax} tested by components from $S - S_1 - S_2$, and for each component of S_1 tested by components from $S_2 - S$ and for each component of S_2 tested by components from $S_1 - S_2$.
(c) An outcome 0 for each component of $S - S_2$ tested by components from $S - S_2$ and for each component of $S - S_1$ tested by components from

$S - S_1$, except for the outcomes given by (a) above.

(d) The remaining outcomes are arbitrary.

Consider another syndrome contructed exactly in the same manner as in the four steps above using S_b in place of S_a and S_{bx} in place of S_{ax} and the arbitrary outcomes given by step (d) are identical. As neither of the two conditions are satisfied, both the two syndromes will have less than x 0^1's. The two syndromes are identical when in each syndrome each 0^1 is replaced by a 0 and these two syndromes are possible syndromes when S_1 and S_2 are respectively the sets of faulty components. Hence, (S_1, S_2) is an indistinguishable pair.

Sufficiency: If condition (i) is satisfied, there exists at least some z, $z \in S_1 - S_2$, such that the outcome of z tested by some component from $S - S_1 - S_2$ is not 0^1. When S_1 is the set of faulty components, this outcome must be 1 in all the syndromes, none of which is identical to the syndromes produced when S_2 is the set of faulty components, as under that fault condition, this outcome would have been 0 in all the syndromes. Hence, S_1 and S_2 is a distinguishable pair. Using the identical argument and interchanging the roles of S_1 and S_2, it may be observed that S_1 and S_2 is a distinguishable pair whenever the condition (ii) is satisfied. This completes the proof.

From Lemma 4, we can formulate the necessary and sufficient conditions for $t[x]$ - diagnosability of a system as follows.

Theorem 3: A system is $t[x]$ - diagnosable, if and only if, for all S_1, S_2, such that $|S_1|$, $|S_2| \le t$ and S_1, $S_2 \subset S$, either of the following conditions is satisfied.

(i) $T(S - S_1 - S_2, S_1 - S_2) \ge x$

(ii) $T(S - S_1 - S_2, S_2 - S_1) \ge x$

Corollary: If a system is $t[x]$ - diagnosable, it is $t[x]$ - diagnosable.

However, given any system, we need not test the validity of Theorem 3 to test for $t[x]$ - diagnosability for all S_1, S_2, such that, $|S_1|$, $|S_2| \le t$. It can be shown that testing the validity of the conditions in Theorem 3 for all S_1, S_2 such that $|S_1| = |S_2| = t$ will be sufficient for the $t[x]$ - diagnosability of a system.

Lemma 5: If a system is $t[x]$ - diagnosable, then every component of the system must be tested by at least $t+x$ other components.

Proof: The proof is very similar to that of Lemma 2. Referring to the proof of Lemma 2, it may be observed that, for S_1, and S_2 defined there,

$S_1 - S_2 = \{v\}$, $S_2 - S_1 = \emptyset$ and $T(S - S_1 - S_2, S_1 - S_2) \le x$. Thus Lemma 5 follows.

Lemma 6: If for every S_1, S_2, such that $|S_1| = |S_2| = t$, and S_1, $S_2 \subset S$, either of the conditions of Theorem 3 is satisfied, then for all S_a, S_b such that $|S_a|$, $|S_b| \le t$ and S_a, $S_b \subset S$, either of the conditions of Theorem 3 is satisfied.

Proof: We will have to consider three cases.

Case I: $S_a \subset S_b$. Proof is similar to that of Case I of Lemma 3. In this case, $S_a - S_b = \emptyset$ and arguing exactly as in Case I of Lemma 3, it can be shown that $T(S - S_a - S_b, S_b - S_a) \ge x + 1$, i.e., at least one of the conditions of Theorem 3 is satisfied by S_a, S_b.

Case II: $S_a \not\subset S_b$ and $|S_a| < |S_b|$. In this case also arguing exactly as in Case II of Lemma 3, it may be observed that,

$S_{a1} - S_{b1} = S_a - S_b$ and $S_b - S_a \supset S_{b1} - S_{a1}$, where S_{a1}, S_{b1} are formed as it was done there. Moreover, $S - S_a - S_b \supset S - S_{a1} - S_{b1}$ and $|S_{a1}| = |S_{b1}| = t$. Thus, if either of the conditions of Theorem 3 is satisfied with S_{a1}, S_{b1}, it will be satisfied for S_a, S_b also.

Case III: $|S_a| = |S_b|$. Using the same argument as in Case II and using S_a instead of S_{a1}, we will have either of the conditions of Theorem 3 satisfied for S_a, S_b also.

Combining the results of Lemma 6 and Theorem 4, we can specify the necessary and sufficient conditions for $t[x]$ - diagnosability as follows.

Theorem 4: A system is $t[x]$ - diagnosable if and only if for all S_1, S_2, such that $|S_1| = |S_2| = t$ and S_1, $S_2 \subset S$, either of the following conditions is satisfied.

(i) $T(S - S_1 - S_2, S_1 - S_2) \ge x$

(ii) $T(S - S_1 - S_2, S_2 - S_1) \ge x$.

Discussion

Necessary and sufficient conditions for the diagnosability of a system with multivalued test outcomes have been presented in this paper. Multivalued test outcomes arise because of either missing values of test result or because of intermittent failure of components. Earlier results in connection with this problem concern with systems in which no two components test each other. A general solution of the problem for any arbitrary system has been presented in this paper.

References

[1] J. T. Butler, "Properties of three-valued system diagnosis," Proc. 11th International Symposium on Multiple Valued Logic, Oklahoma City, Oklahoma, 1981, pp. 85-89.

[2] F. P. Preparata, G. Metze and R. T. Chien, "On the connection assignment problem of diagnosable system," IEEE Trans. Electronic Computers, Vol. EC-16, pp. 848-854, 1967.

[3] J. T. Butler, "A three-valued system diagnosis model which accommodates intermittent faulty processors," Submitted to 13th International Symposium on Multiple Valued Logic, Kyoto, Japan, 1983.

[4] S. Mallela and G. M. Masson, "Diagnosis without repair for hybrid fault situation," IEEE Trans. Computers, Vol. C-29, pp. 461-470, 1980.

[5] S. Mallela and G. M. Masson, "Diagnosable systems for intermittent faults," IEEE Trans. Computers, Vol. C-27, pp. 560-566, 1978.

[6] A. Sen Gupta, A. Sen and S. Bandyopadhyay, "On system diagnosis in presence of intermittent faults," Submitted to IEEE Trans. Computers.

A METHOD OF TEST GENERATION
FOR VERIFICATION OF WIRING CORRECTNESS

Krzysztof Bucholc

Technical University of Poznan, Regional
Computer Center, 60-965 Poznan, Poland

Abstract

The paper presents a method of test
generation for any multiterminal wiring
network, such as printed circuit board,
computer backpanel wiring etc. The method
is based on the minimization of the test
generated by examining the correct network
with computer-controlled tester. Three-
valued algebra was used. In comparison
with more straightforward algorithm based
on the adjacency matrix, lower space com-
plexity has been achieved - $O(n)$ rather
than $O(n^2)$.

Introduction

In comparison with subassembly testing,
wiring testing is based on a very simple
class of fault types. It involves shorts
and opens.

If we call the set of terminals con-
nected together a cluster, then the wiring
testing can be defined as a process which
determines the presence of connection be-
tween any pair of terminals in a cluster
and the absence of shorts between any pair
of terminals from two different clusters.
The number of terminals in computer back-
panel wiring network implies the import-
ance of the problem. For the same reason
low time and space complexity of test
generation and testing algorithms is very
important.

The test for a wiring network can be
presented in form of a sequence of
triplets (p,k,v) where p,k are labels of
the terminals and value of v indicates
presence or absence of connection between
p and k in correct network - 1 and 0,
respectively.

A convenient way to obtain the test
is to generate it automatically from the
correct network. The most straightforward
algorithm is as follows: Each pair of ter-
minals is checked for connection. Labels
of the terminals and the result of this
elementary test are stored. The sequence
of all $n(n-1)/2$ triplets is the required
test.

The test obtained in this way is
highly redundant. Minimization can essen-
tially reduce the volume of required stor-
age and test execution time.

Test minimization - introduction

Let us denote the number of terminals
in the network by n , the number of clus-
ters by q and the number of single
terminals by p . Testing for shorts re-
quires[1,2]

$$T_s = q(q-1)/2$$

tests.
Testing for opens requires[2]

$$T_o = n-q$$

tests.
The total number of tests is

$$T = T_s + T_o = q(q-1)/2 + n - q$$

The non-minimized test set consists of

$$G = n(n-1)/2$$

elementary tests. The number of redundant
elementary tests is equal to

$$R = G - T = n(n-3)/2 - q(q-3)/2$$

Except for the network consisting of
single terminals $q<n$ and $R>0$

Example 1

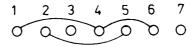Let us consider the network shown
in Fig. 1.

Fig. 1. Considered network

Tests $(1,4,1)$, $(4,6,1)$, $(2,5,1)$ check
connection between pairs of terminals
which must be connected. Tests $(1,2,0)$,

(1,3,0), (1,7,0), (2,3,0), (2,7,0), (3,7,0) check the absence of connection between terminals from two different clusters (cluster may consist of only one terminal). The 9 elementary tests, out of 21 possible, constitute the minimal test for this network.

Very often single terminals i.e. clusters consisting of only one terminal are of no practical importance and may be not included in the test. It leads to a further minimisation.
Testing for shorts in this case requires

$$T_s' = (q-p)(q-p-1)/2 \qquad \text{tests.}$$

Testing for opens requires

$$T_o' = T_o = n-q \qquad \text{tests.}$$

The total number of tests is

$$T' = (q-p)(q-p-1)/2 + n-q$$

and the number of redundant tests

$$R' = G - T' = n(n-3)/2 - q(q-3)/2 - p(p+1)/2 + pq$$

Example 2

For the network shown in Fig. 1 four elementary tests (1,2,0), (1,4,1), (4,6,1), (2,5,1) suffice, when single terminals are excluded.

Example 3

The number of tests before minimization G and after minimization T and the percentage of redundant tests, for a few backpanel networks, are listed in Table 1.

Table 1

n	q	p	G	T	$\dfrac{G-T}{G}$ %
3927	1853	1157	7708701	243934	96.8
4125	2527	1208	8505750	870819	89.8
924	825	765	426426	1770	99.6

The method of test minimization

To detect shorts it is sufficient to check connection between selected terminals - one from each cluster. Similarly to detect opens between terminals which belong to the same cluster c-1 tests, out of c(c-1)/2 possible, suffice, where c is the number of terminals in the cluster.

Let us choose one terminal from each cluster, except for single terminals, and mark it. To detect all opens in the cluster it is sufficient to execute tests comprising the marked terminal. There are c-1 such tests. To detect shorts between multiterminal clusters, checking connection between marked terminals is sufficient.

The generation of a minimal test set is performed in two stages. At the beginning all possible two-terminal tests are executed. Results are stored. Simultaneously the set of terminals is split into three subsets - single terminals, marked terminals and ordinary terminals (terminals which belong to the cluster consisting of at least two terminals and are not marked). To each terminal is assigned a variable which value indicates the state of the terminal (the group to which the terminal belongs). The codifying is as follows:

  0 – ordinary terminal in a multiterminal cluster,

  1 – single terminal,

  2 – marked terminal.

In the second stage the test is minimised.

Definition 1. Operation P

Let us denote the set of all terminals by K, and the set of test results by V; $V = \{0,1\}$, where 1 means connection, 0 absence of connection. With every terminal there is associated a variable denoted s(k), where $k \in K$, which takes values 0, 1 or 2.

We shall define the operation P as a mapping

$$(v, s(p), s(q)) \rightarrow (s(p), s(q))$$

where $v \in V$, p, q $\in$ K, and

$s(p) = s(p)$ if   v=0

$s(p) = f_1(s(p), s(q))$ if   v=1

$s(q) = s(q)$ if   v=0

$s(q) = f_2(s(p), s(q))$ if   v=1

Functions f_1 and f_2 of three-valued variables are defined in Table 2 and Table 3 respectively.

Table 2

x_1	x_2	$f_1(x_1, x_2)$
0	0	0
0	1	0
0	2	0
1	0	1
1	1	2
1	2	0
2	0	2
2	1	2
2	2	2

Table 3

x_1	x_2	$f_2(x_1, x_2)$
0	0	0
0	1	1
0	2	2
1	0	0
1	1	0
1	2	2
2	0	0
2	1	0
2	2	0

Algorithm 1. Determining the state of terminal

1. Assign to every terminal state 1.
2. To every pair of terminals apply P operation.

Theorem 1

After accomplishing algorithm 1, the state of each single terminal is equal to 1.

Proof

For a single terminal, result of every elementary test is 0 - no connection. For v=0 P operation does not change states of terminals. As at the beginning all terminals received state 1, after accomplishing of algorithm 1 the state of single terminal remains equal to 1.

Theorem 2

After accomplishing algorithm 1, exactly one terminal in every multiterminal cluster has state 2, states of other terminals in this cluster are 0.

Proof

First we shall prove that in every multiterminal cluster there exists at least one terminal which state is 2.

Let us consider one cluster. In the beginning the states of all terminals are 1. The first P operation applied to this cluster creates states 2 and 0 /by Def.1/ Once created state 2 cannot cease to exist - there is no such P operation, that if v=1 and at least one argument is 2, the resulting states both differ from 2.

Now we shall prove that there is exactly one such terminal. Let us asume that there are two such terminals. It implies that neither of them has been in state 0 at any stage of algorithm 1 execution. /This comes straight from Def. 1 - there is no operation which changes state 0 into another state./ This leads to the conclusion that P operation has not been applied to the considered pair of terminals - if v=1 there is no such P operation that states of both terminals after the operation differ from 0. This contradicts our assumption that P is applied to every pair of terminals.

Finally we shall prove that none of the terminals is in state 1. Let us assume that there is a terminal which state is 1. The state of terminal after P operation remains equal to 1 if and only if the state of the other terminal is 0 /see tables 2 and 3/. As the P operation is applied to every pair of the terminals, our assumption implies that the states of all terminals, except for the terminal under consideration, are equal to 0. It contradicts the previous conclusion that there is at least one terminal which state is equal to 2.

At the stage of minimization, some tests are selected to obtain the minimal set of tests. Let us denote this set by M.

Algorithm 2. Test minimization

1. Take the first stored test.
2. If test result v equals to 0 and states of considered pair of terminals are equal to 1 or 2, then include the test in M. If v=1, the state of one terminal is 2 and the state of the other terminal is 0, then include the test in M. In all other cases do not include the test in M.
3. If not end of file, take the next test and go to 2.

If single terminals are not to be tested, only tests in which both terminals are in state 2 are taken when v=0.

These rules are also presented in another form in tables 4 and 5. Table 4 refers to the case when single terminals are to be tested. Tests which belong to the minimal set of tests are designated with "+". Combinations which do not appear in practice are marked "N". Table 5 is used when single terminals are not tested.

Table 4			
v	s p	s q	
0	0	0	
0	0	1	
0	0	2	
0	1	0	
0	1	1	+
0	1	2	+
0	2	0	
0	2	1	+
0	2	2	+
1	0	0	
1	0	1	N
1	0	2	+
1	1	0	N
1	1	1	N
1	1	2	N
1	2	0	+
1	2	1	N
1	2	2	N

Table 5			
v	s p	s q	
0	0	0	
0	0	1	
0	0	2	
0	1	0	
0	1	1	
0	1	2	
0	2	0	
0	2	1	
0	2	2	+
1	0	0	
1	0	1	N
1	0	2	+
1	1	0	N
1	1	1	N
1	1	2	N
1	2	0	+
1	2	1	N
1	2	2	N

The whole process of test generation is illustrated in Example 4.

Example 4

Let us consider the network shown in Fig. 1. We shall trace the whole process of test generation. In the beginning the states of all terminals are 1 /see Table 6/ In the first stage, for each pair of terminals we execute elementary test and apply P operation. Test results are shown in Table 6. If the test result is 1 /presence of connection/ the operation P can change the states of considered pair of terminals. In this case the states of all terminals are shown in Table 6.

In the second stage we minimise the test. If single terminals are to be tested we select tests according to Table 4, otherwise we use Table 5. Final results are as follows:

Test set comprising single terminals: (1,2,0), (1,3,0), (1,4,1), (1,6,1), (1,7,0), (2,3,0), (2,5,1), (2,7,0), (3,7,0).

Test set if single terminals are not to be tested: (1,2,0), (1,4,1), (1,6,1), (2,5,1).

RAM is required, what leads to the space complexity of $O(n^2)$.

In both cases large sequential access memory is required. The volume of required storage is much smaller if at the first stage test results are not stored. In this case, at the second stage all the elementary tests must be executed once more.

References

1. Kautz W.H., Testing for Faults in Wiring Networks, IEEE Trans. on Comput., C-23,4, pp. 358-363.

2. Bucholc K., Rajski J., Testing algorithms for large wiring networks, Proc. of 3rd Intern. Conf. on Fault-Tolerant Systems and Diagnostics FTSD-80, Katowice 1980, pp. 91-96.

3. Aho A.V., Hopcroft J.E., Ullman J.D., The Design and Analysis of Computer Algorithms, Addison Wesley, Reading Mass., 1974.

Table 6

Test			Terminal state						
p	q	v	1	2	3	4	5	6	7
			1	1	1	1	1	1	1
1	2	0							
1	3	0							
1	4	1	2	1	1	0	1	1	1
1	5	0							
1	6	1	2	1	1	0	1	0	1
1	7	0							
2	3	0							
2	4	0							
2	5	1	2	2	1	0	0	0	1
2	6	0							
2	7	0							
3	4	0							
3	5	0							
3	6	0							
3	7	0							
4	5	0							
4	6	1							
4	7	0							
5	6	0							
5	7	0							
6	7	0							
			2	2	1	0	0	0	1

Conclusions

A method of test generation for verification of wiring correctness has been described. The algorithm requires 2 bits of random access memory per terminal. It gives the space complexity $O(n)$. For more straightforward algorithm, based on the adjacency matrix, $n(n-1)/2$ bits of

Session 8B
Logic Design III

AUTOMATED DESIGN OF COMBINATIONAL NETWORKS UNDER SPECIFIC CONSTRAINTS: A THEOREM PROVING APPROACH[†]

Waldo C. Kabat[‡]
Northwestern University[‡]
Evanston, Illinois 60201

ABSTRACT

An automated theorem proving system is seen as a viable addition to the set of traditional design automation tools. The automated design of combinational network for an arbitrary switching function can be performed using theorem proving techniques. Additional constraints such as modularity, design under the requirements of a particular technology, and the fault tolerant logic design can be imposed upon the design.
Index Terms — Logic design, automated theorem proving, multi-valued logic, design automation.

I. INTRODUCTION

The modern logic design is a task of high complexity and inevitably involves the automation of the design process. The use of a theorem proving system may be a valuable addition to traditional automation tools. The method for an automated synthesis of combinational logic that was presented in [13,14], is based entirely on theorem proving techniques, and can be used to perform design in any *multi-valued system (including binary)* using an arbitrary, functionally set of connectives. What was not shown, however, is that the method easily extends to the design under a *variety of constraints*. For instance, constrains such as modularity, design under requirements of a particular technology, and the consideration of the fault tolerant design via internal redundancy with complementary logic, can be imposed.

As building blocks grow more complex and highly functional [10,17,22,27], and high level logic primitives are used to simplify both circuit modeling and test generation [3,4], the formalism of modular design needs to be develo-

ped. Section 2.2 presents the theorem proving approach to the problem.

Rapidly evolving technologies (e.g., CMOS, NMOS, Bipolar etc.) require that the design tools can either effectively accommodate new technology design rules or have some measure of technology independence. Ultimately, the rule generation process should be formalized to bridge different and changing technologies such that the design integrity be improved and manual rule generation eliminated. Section 2.3 shows that the use of a theorem proving system in the design process offers some answers to these issues. The I^2L technology is used as an example to illustrate the approach. Both, an accommodation of new design rules and a technology independence may be possible. Moreover, the presented approach consists of a totally general solution for an arbitrary switching function that is to be implemented in I^2L.

The internal redundancy with complementary logic is one of the standard VLSI techniques employed to achieve a fault tolerant characteristic of the system [24]. An extension of this, essentially, binary technique to an arbitrary, multi-valued case, is presented in Section 2.4 using theorem proving techniques.

All experiments that are presented in this paper were performed on the general purpose, first-order logic, resolution based theorem proving system [18,25]. Finally, it is assumed that the reader understands basic concepts of theorem proving.

II. AUTOMATED DESIGN UNDER SPECIFIC CONSTRAINTS.

In [13,14] we presented an approach to automation

of the design of combinational logic. The heuristic method that was shown consists of a theorem proving implementation of a systematic, uniform procedure for the synthesis of an arbitrary switching function. Any multi-valued (including binary) function can be synthesized in a top-down fashion using the functional blocks of the designer's choice. Brief description of the method is included here to make the presentation self-contained.

2.1 The Method for Automated Synthesis of Combinational Logic.

The method is based on the axiomatization of the logic design environment, that is the logic system, the logic connectives, and the rules for obtaining an acceptable solution [29,30]. The steps of the method are: input of the function truth table or tables corresponding to all permutations of input variables, derivation of a canonical circuit structure using desired building blocks, simplification of the obtained circuit structure using the properties of the blocks and final circuit synthesis. Figure 1 illustrates the process of an automated synthesis of the logic circuit on the example of one trit of 2×2 Ternary Multiplier:

In step 1 a function is presented to the theorem prover as a target clause which is a one to one mapping from a function truth table to the clause form. Note that the function $T(V,X0,X1,X2)$ represents a collection of entries from the truth table corresponding to the range of the ternary variable V, and functions MULT and TRITO denote the multiplier and the least significant trit of the product, respectively. Optionally, if the "best" logic circuit is of concern, remaining input target clauses corresponding to all permutations of input variables of the function can be generated.

Step 2 results in the derivation of a canonical circuit structure for the function in question. The circuit structure is called canonical because it represents the canonical form of a switching function, and is derived using a building block of the designer's choice; example dual 4-input multiplexer as shown in Figure 1.

The simplification of the target clause representing a canonical circuit structure occurs in step 3. Simplification is performed by rudimentary simplifiers, that is, demodulators corresponding to the basic properties of a chosen building block. As a result the fully simplified target clause is generated. It can be shown

[15] that there is mathematical relationship between the demodulation simplification process and the rules for simplifying switching expressions.

The last step results in the final circuit synthesis, or the derivation of an empty clause. The "best" clause is selected based on its weight attribute. Intuitively, the lightest clause is the one which resulted from a massive application of simplifiers and, indeed, is selected for further synthesis. The synthesis algorithm, described in [30], provides for the recovering of a circuit structure from the trace of the proof from the empty clause to the input target clause.

Demodulation [31] is a primary theorem proving technique which performs the steps of the method. It is used in two capacities: as simplification and as canonicalization. Hyperresolution is used as a chief inference rule because it can produce powerful inferences in a single step. The general strategy is that of Set-of-Support [5,32] which restricts the inferences to those which are relevant to the synthesis of the function. Furthermore, it is coupled with demodulation to reduce the number of clauses that the theorem prover retains during its search for a proof.

It is seldom a case that just a simple logic structure is being sought in the design. Additional constraints, such as modularity, testability, maintainability, technology dependence, fault tolerance, and others, are often imposed upon the design. For the design to be stable, small modifications of system function must result in small modifications of the design. For the design procedure to be stable, small modifications of the design specification should result in small modifications or no modifications of the procedure. It is important that the imposition of additional constraints on the design should leave the method for logic synthesis stable. Modular design, technology dependent design, and consideration of the fault tolerant logic design are further discussed in the following sections.

2.2 Modular design.

Recent projections [17,22,27] indicate that future building blocks will definitely be more complex and highly

367

functional. It is also conceivable that high level primitive logic elements are to simplify both circuit modeling and test generation [3,4]. Taking the above into consideration, one must find a formalism different than, for instance, that of Boolean and Post algebras. Although powerful and well developed, algebraic descriptions become of less benefit as the level of abstraction of the system design increases. At the logic design level, the formalisms should appear as structures and rules rather than mathematical constructs [28]. The set of well defined structures and rules for interconnecting them helps to establish systematic procedures for logic design. A well defined structure can be a building block or a module which the designer selected, and the structure interconnection rules can be rules of module interconnections. Dependent upon no particular technology, a module, or a set of modules, which is functionally complete is used in the course of a flexible, disciplined and correct design process.

The method for the synthesis of combinational logic that was described in [13,14] is based on the properties of a predefined building block. The building blocks used were the general multi-valued T-gate and the multi-valued multiplexer, Figure 2 a) and b), respectively. The multiplexer is a standard component in commercial digital design and is used further to illustrate our approach. Since we are concerned with the formalism of modular design for an arbitrary m-variable, R-ary switching function, the "m-ary R^m-input R-ary" multiplexer will be used as a module in the design. The explanation of this terminology follows. Also, if one is to be bound by the requirement of a specific technology, the multiplexer is a known building block which can be implemented to perform in different logic systems. I^2L technology is a good example here, and is discussed in the next section.

Different types of multiplexers can be defined based on the logic system in which the multiplexer is to perform. Variations occur, for example, in the types of control inputs available (double-rail or single rail), and in the interconnections between terminals and other multiplexers. The most general form of the MUX is defined by the triple $\Omega = (\mathbf{X},\mathbf{V},\text{OUT})$, where $\mathbf{X} = \{X_1,...,X_{R^m}\}$ is a set of R-valued MUX inputs, $\mathbf{V} = \{V_1,...,V_m\}$ is a set of R-valued control inputs, and $\text{OUT} = \{0,1,...,U\}$ is an R-valued MUX output. Furthermore, any control signal and any input are bound by $V_i \in \{0,1,...,U\}$ and $X_j \in \{0,1,...,U,V_i,-V_i,RF,\text{OUT}\}$, respectively. Thus, any legitimate constant, a double rail control variable, V_i, a residue function, RF, or an output of another multiplexer are accepted as inputs. The residue function is a trivial function, such as an arithmetic plus. The relation between Radix and Unit is: $U = R-1$. A pictorial representation of the Ω is shown in Figure 2 b).

To show how the multiplexer module and its interconnection rules are implemented using theorem proving techniques, let us consider the standard 4-input binary multiplexer. The formal definition and corresponding graphical symbol are shown in Figure 3 a). The decomposition axiom Figure 3 b), not only reflects the structure of the MUX, it also embeds additional information by means of functions and function variables which are used. Whenever used, the axiom (formula) is interpreted and its interpretation carries a structure and an assignment of values to functions and function variables of the formula. The structure of the MUX as a logic device is reflected in the structure of the formula. As written, Figure 3 b), the formula for the MUX has four inputs, X0,X1,X2,X3, the selection part, SEL(V1,V2), connected to the device MUX, and the MUX(SEL().) as an output. The MUX structure consisting of the inputs, selection part and output is constructible which is stated by the predicate CKT(MUX(SEL().)). The MUX decomposition axiom is written in the clause form and represents the following implication:

CKT(SEL(V1,V2))∩CKT(X0)∩CKT(X1)∩CKT(X2) •
CKT(MUX(SEL(V1,V2),X0,X1,X2,X3));

To say it differently, given the elements of the MUX, the construction of the MUX itself is imminent

To be a part of a logic network, the MUX must conform to its specification, for instance, permitted input and output signals are 0 and 1. The adherence to the specification is enforced by the assignment of values to functions and function variables of the MUX formula. The X_i variable denotes the situation in which any well defined input can be accepted. The V_j variable (an argument of SEL) and the output MUX(SEL(),) have their scopes determined by the assignment of the formula as well. An assignment changes from the initial one as stated in the input target clause, through the intermediate ones corresponding to the transformations of the target clause, to the final assignment as determined in the fully simplified clause. Transformations that occur in the course of a proof are caused by subsequent resolutions and demodulation on any intermediate target clause. Any change in the assignment that might result during a resolution, for instance, caused by unification, conforms to the MUX specification. Both as simplification and canonicalization, demodulation causes a well defined, finite transformation of any given clause; and by choosing a specific set of demodulators, one has full control over possible changes made to the assignment. For the 4-input binary MUX, this set of demodulators-simplifiers is shown in Figure 3 c).

Module interconnections are taken care of in the following way. The input target clause represents a one to one mapping from the function truth table to the clause form. It is transformed into the structure which represents a canonical logic circuit for the function, using the dual 4-input MUX. A canonical logic circuit is a legal interconnection of modules which is simplified using properties of the module.

Therefore the simplification is performed according to the rules of module interconnections. A fully simplified target clause is decomposed using the decomposition clause for the module, Figure 3 a), again adhering to the rules for module interconnections.

In the example of the Seven-Segment Display binary function [1], Figure 4 a), the target clause representing a canonical circuit structure, shown in Figure 4 b), is demodulated to the clause in Figure 4 c). As can be seen, no rule for the scope of input variables, control variables, or rules for module interconnections were violated. Another example is that of the ternary 9-input multiplexer, as defined in Figure 5 a-c), and used in the design of the logic circuit for a 2×2 Ternary Multiplier, Figure 6 a). The target clause representing a canonical circuit for the 2×2 Ternary Multiplier as shown in Figure 6 b), is demodulated to the clause in Figure 6 c). Modular logic circuits for the Seven-Segment Display and for the 2×2 Ternary Multiplier functions that were obtained by the theorem prover, are shown in Figures 4 d) and 6 d), respectively.

It can be seen that using the formalism of a modular design, an arbitrary m-variable, R-ary function can be synthesized.

2.3 Design under Technology Requirements.

So, we have just completed the synthesis of a function given a required building block. and we would like to know if it can be implemented under requirements of a particular technology. To begin with, one must decide on the selection of technology. The choice is influenced by the current availability of multi-valued circuits that were developed for this technology. Multi-valued circuits developed over the last decade can be grouped into three major classes [8,26]: the Bipolar current mode circuits which comprise I^2L and ECL quaternary logic, the unipolar

voltage mode circuits including NMOS and MESFET ternary logic, and the charge-coupled quaternary logic. By far the first class is best understood. It adopts many techniques of the current mode of signal processing and is general as well as flexible, for it stems from the synthesis of threshold logic. It is not limited to any specific radix, but for practical reasons, only binary, ternary, and quaternary logic circuits have been designed so far. The second class is practically confined to ternary logic, while the third one, although initially successful, is not economically viable as yet. Therefore we will use as an example I^2L technology.

Until now, the most comprehensive approach to the design of multi-valued I^2L logic circuits was due to McCluskey [19,20,21]. It is conceded there that the algebraic system geared to I^2L technology, and the set of operators which are implied by the system and selected based on a judicious study of already designed circuits, are responsible for the efficient designs of multi-valued combinational logic that have been obtained. The design procedure relies on *the modified Karnaugh map, Q-Map, and the universal quad gate*. A slightly modified version of this approach was implemented on the automated theorem proving system [29].

While the Karnaugh map approach of McCluskey can be seen to be of historical significance, the theorem proving approach looks more promising when weighted against the complexity of today's systems. This is especially so given the recent interest in nonstandard logic. Due to the observation made by VLSI designers on the role of circuit minimization in large binary logic designs [11], multi-valued logic building blocks may be seen to provide a means to limit the scope over which circuit minimization must range. Such blocks of locally optimized, general purpose logic could reduce the interconnection burden. Since, on the average, 70% of the chip area is consumed by wiring, while the remainder is occupied by active devices and by passive isolation

[8], one can easily see why this is of great interest to the logic designer.

Our method for the design of I^2L combinational logic consists of a two phase process

1) Function synthesis using a given building block

2) End optimization at the unary block level

The first phase is performed using techniques of the method for automated synthesis of combinational logic [14,15], and in the end, corresponds to the cascading of I^2L building blocks. The second one performs the local optimization of the logic remnants from the first phase

I^2L technology can be briefly characterized as follows [6,7]: i) operation in current mode with a fixed number of current levels with each current level representing a logic level, ii) replication of current signals by means of a current-mirror as shown in Figure 7 a) with implied conversion of the source current in the base into the sink current in the collector, and with a limit on the number of replicated copies of four, iii) linear summation provided by connecting together leads carrying current mode signals, Figure 7 b), iv) an easy way of integration of constant current sources into the gate structure by means of thresholding, v) the presence of a switch, a transistor operating in normal mode and providing an infinite sink, if a positive source is applied to its base, Figure 7 c), vi) availability of a complement gate performing the operation of multi-valued negation, Figure 7 d). As will be seen from the following discussion, we adhere to the requirements of I^2L technology while both phases of the method are being carried out. Meantime, as I^2L technology continues to develop, these further developments can be incorporated into our approach.

First phase: *Function Synthesis Using Given I^2L Building*

Block.

Four steps of the method for the synthesis of combinational logic as described in [14,15] are to be executed during this phase. The I^2L multiplexer is a basic building block used in the process of logic synthesis. We also introduce a minor modification in the final step to ensure that the decomposition stops at the unary building block (vector) level, to be described in the sequel.

The axiom

CL -I2L(X) -I2L(V0) -I2L(V1) -I2L(V2) -I2L(V3)
    I2L(T(X,V0,V1,V2,V3));

   where:
      I2L - predicate,
      T   - quaternary T-gate function,
      X   - control variable,
      Vi   - placeholder for another T-gate, constant, etc.

represents the four-valued I^2L multiplexer, or a T-gate, as presented in [6], Figure 8. As a propositional formula it carries an induced interpretation, that is, it has a structure and an assignment. The structure of the formula corresponds to the structure of an I^2L quaternary multiplexer in that the X is a control variable and the Vi's are input variables, and the ties between them are such that only one Vi is selected depending upon the value of X. It is also assumed that the additional I^2L technology considerations are taken care of, such as, a current limit of approximately four on the number of collectors that a transistor can have, reflected in the circuit for the five-valued MUX, although the corresponding axiom is not that much different from the four-valued one, Figure 9. Furthermore, the X as well as the Vi's are only source type signals because of the way the multiplexer clause is used in the course of a proof. The assignment of values to X and Vi's will force this to happen.

Since the multiplexer decomposition is to stop at the unary block level, the actual decomposition clause must have the following form:

CL I2L(T(X,Y1,Y2,Y3,T(Z,V0,V1,V2,V3)))
    -I2L(X) -I2L(Y1) -I2L(Y2) -I2L(Y3) -I2L(T(Z,V0,V1,V2,V3));
CL I2L(T(X,Y1,Y2,T(Z,V0,V1,V2,V3),Y3))
    -I2L(X) -I2L(Y1) -I2L(Y2) -I2L(Y3) -I2L(T(Z,V0,V1,V2,V3));
CL I2L(T(X,Y1,T(Z,V0,V1,V2,V3),Y2,Y3))
    -I2L(X) -I2L(Y1) -I2L(Y2) -I2L(Y3) -I2L(T(Z,V0,V1,V2,V3));
CL I2L(T(X,T(Z,V0,V1,V2,V3),Y1,Y2,Y3))
    -I2L(X) -I2L(Y1) -I2L(Y2) -I2L(Y3) -I2L(T(Z,V0,V1,V2,V3));

That is to say, the decomposition will proceed if and only if the term T() contains at least another term T() within itself. The corresponding logic structure is a two level one, with the bottom level being the unary block level as the decomposition comes to an end.

Let us consider the example of a Quaternary Full Adder, Figure 10. The final clause obtained in the first phase will be the following:

CL I2L(T(D0,C1,C2,C3,C0)) I2L(T(D0,C2,C3,C0,C1))
    I2L(T(D0,C3,C0,C1,C2)) I2L(T(D0,C0,C0,C0,C1))
    I2L(T(D0,C0,C0,C1,C1)) I2L(T(D0,C0,C1,C1,C1));

   where:
      D0   - control variable,
      Ci   - constant i.

The literals of this clause denote the composite unary functions. Needless to say, if one stops the synthesis at this point, the unary functions would default to be implemented using quaternary multiplexers, Figure 15 a). Otherwise, local optimization is performed as the second phase.

Second phase: *End Optimization at Unary Block Level.*

Having simplification carried out using the properties of a given building block, it is worthwhile to optimize the "unsynthesizable" composite functions. This is particularly true for functions which cannot be realized with a small amount of logic or for functions for which the simplification does not yield a simple logic structure. As a result, the end optimization is advocated to produce savings on the number of components, such as transistors, switches, collectors, etc., without sacrifying the regularity of the overall implementation.

The end optimization phase is implemented based on

the axiomatization of the I^2L connectives as described in [30]. Once the building blocks, here the multi-valued I^2L gates, are in place and the set of I^2L primitives are available, the optimization phase can start

To begin with, one needs to choose the set of gates. Following McCluskey's claim of "gate-technology" suitability [21], the PLUS, INHibit and DIFFerence connectives were selected. McCluskey correctly notes that any unary function can always be realized without using the MAX connective [21]. In fact, the realization with PLUS is preferred, since it uses fewer collectors. As a result, the PLUS, INHibit and DIFFerence are the only connectives we use. The following subsection contains material on the axiomatization of the I^2L gates that was presented in [30]

Axiomatization of I^2L connectives: The input-output characteristic of every gate is represented by the matrices of equalities, as shown in Figure 11, which are nothing more than mappings from the function truth tables which a particular gate implements to its clause form. It should be noted that two types of signals are being used: source signals denoted by a constants C_i, and sink type signals denoted by a constants N_i. Furthermore, the PLUS gate accepts only source and outputs sink type signals, the INHibit's output, X, is shorted to the ground when the input current C_i is greater than zero, and the DIFFerence expresses the conversion of signals from sink to source where the sink signal is a mirror of the input source. Undefined values for some input combinations are intentionally left out so that a gate can not produce an output if these inputs occurred

The gate behavior in the design environment is formally defined by the axioms in Figure 12. The PLUS and INHibit are each a two-input, one-output type of gate, and, therefore, they communicate with the outside world by means of the two input vectors, the I2L(UBB(X0,X1,X2,X3)) and I2L(UBB(Y0,Y1,Y2,Y2)) literals, and the one output vector, the I2L(UBB(Z0,Z1,Z2,Z3)) literal. The DIFFerence is a one-input, one-output type of gate, and thus it accepts an input vector, the I2L(UBB(Y0,Y1,Y2,Y3)) literal, and produces an output vector the I2L(UBB(Z0,Z1,Z2,Z3)) literal. Since the process of communication must meet the gate specifications, adherence to this specification is enforced by means of a table lookup for allowable input and output signals, via EQU literals. The table lookup is performed whenever the axiom is used, and the resolvent clause represents the permitted gate output provided the unification on the output literal and the gate specifications literals could have occurred. Successful unification is recognized when no EQU literals are present in the resolvent clause

Lexicon of I^2L primitives: It is a typical design practice to have the set of primitive elements always available when designing complex ones. In our case the lexicon of primitive I^2L components consists of all of McCluskey's strong threshold and delta literals [21] conveniently generated by the circuit in Figure 13. The set of demodulators corresponding to these primitives is shown in Figure 14. Note that the variable X can assume any value for any allowable source type of signal. As soon as the newly generated clause contains a literal which matches any one of the primitives, it is demodulated to a harmless function, NTH(A), to denote the fact that this primitive has just been used at this stage of a design. A demodulator's number indicates which primitive was used.

To continue with the example of a Quaternary Full Adder, we show the circuits for the composite unary functions of the Sum and the Carry. Outputs of these circuits are inputs to the multiplexer cascade obtained in the previous phase. Figure 15 a,b) displays the I^2L implementation of a quaternary full adder obtained by the theorem prover. As a further example, consider the three variable ternary func-

tion [16], Figure 16 a), and the one stage of a quaternary ALU [12], Figure 17 a-e). Corresponding I^2L circuits generated by the theorem prover are shown in Figure 16 b) and Figure 17 f-g).

As can be seen, the method consists of a totally general solution for an arbitrary function that is to be implemented in I^2L technology. The synthesis of a unary function is reduced to a trivial case and involves only the end optimization phase. Clear advantage of this method is the hierarchical design which can be easily modified to accommodate changes in I^2L technology and functionality. We believe that logic design in different technologies can be performed in a similar fashion using the method. However, compared with other methods for the design of I^2L circuits [6,9], our method does not generate optimal solutions.

2.4 Fault Tolerant Logic Design.

In this section we consider a theorem proving implementation of a specific technique for logic design, namely, the use of internal redundancy with "complementary logic" which is used in the design of fault-tolerant systems [2,23,24]. Our objective is to design a fault tolerant combinational network for the given function.

With the advent of VLSI it has become clear that the practice of externally duplicating logic and comparing it is neither practical nor useful [24]. The use of internal redundancy is much preferred. Duplication with complementary logic shows that identical failure states can be avoided and the design made less susceptible to bridging faults. The two outputs of the functional and complementary circuits are later compared for correctness of operation. It is important to note, that for the general case of a multi-valued system the notion of complementary logic actually refers to the logic circuit for which all control variables are negated, yet the circuit performs the original operation. The theorem prov-

ing implementation of this technique generates logic circuits for the functional and complementary combinational logic for an arbitrary switching function at the same time. Basically, four steps of the method for the synthesis of combinational logic are used. The chief modification involves the step for the function input.

Let us use a three variable ternary function, Figure 18, as an example. Note that the notion of duplication is expressed by simply repeating the function table within the encapsulating function EX (example function). Since the complementary circuit is to compute the same function value for the same combination of input variables and yet the input variables are provided in the complementary form, the corresponding function truth table needs to be prepared. The set of demodulators to perform this task for the sample function is shown in Figure 19. Demodulation is triggered by hyperresolution, LIST BACKWARD, and results in a clause containing two forms of the same function, the functional and its complement. Starting from this point, the rest of the synthesis process continues as before. The generation of all table permutations, if any, the derivation of a canonical circuit structure, the simplification and the final synthesis steps are performed simultaneously on the functional and complementary forms of the target clause. As a result, the functional and complementary logic circuits are generated. Figure 20 shows the corresponding logic circuit for the sample function. As another example, consider the ternary full adder function. Corresponding functional and complementary logic circuits generated by the theorem prover are shown in Figure 21.

III. SUMMARY AND CONCLUSIONS.

We have described how additional constraints, such as modularity, technology dependence and fault tolerant logic design can be imposed upon the design of the combi-

national network for an arbitrary switching function. The method for an automated sythesis of combinational logic [14] is used as a basis for the constrained design. The formalism of modular design of an arbitrary m-variable, R-ary switching function is demonstrated using a Universal Logic Module which provides a general logic structure, rules for structure interconnections and the structure behavior. Technology design at the transistor level is shown with the example of I^2L technology. The design process consists of two phases: 1) the function synthesis using a building block pertinent to the technology and 2) the end optimization at the block level. The method for an automated logic synthesis is good for the first phase, while the second phase involves the optimization of "unsynthesizable" composite functions by using primitives and gates inherent in the technology. I^2L primitives are McCluskey's [21] strong and delta literals. The PLUS, DIFFerence and INHibit operators are the only gates used in the second phase of a design. Fault tolerant logic design is carried out via the approach of internal redundancy with "complementary" logic. Once a complementary truth table for the given function is created, both the functional and complementary tables are inputs to the method for logic synthesis. A fault tolerant logic circuit for an arbitrary switching function, which consists of functional and complementary circuits, is generated by the method.

The results that were obtained are encouraging. The crucial test of feasibility of the approach has been demonstrated, and an indication of the practicality provided in terms of results from the theorem proving experiments, Table 1. It is expected that these early results will provide a platform for the theorem proving solutions to other problems such as a) the general layout problem, and b) the automatic test generation problem.

VI. ACKNOWLEDGEMENT

REFERENCES

[1] Blakeslee, T. R., *Digital Design with Standard MSI and LSI*, John Wiley and Sons, 1975.

[2] Breuer, M. A. and Friedman A. D., *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press Inc., 1976.

[3] Breuer, M. A. and Friedman A. D., "Functional Level Primitives in Test Generation", *IEEE Trans. Comput.*, vol. C-29, pp.223-235, March 1980.

[4] Breuer, M. A., Friedman A. D. and Iosupovicz A., "A Survey of the State of the Art of Design Automation", *Computer*, pp.58-75, Oct. 1981.

[5] Chang, C. and Lee R. C., *Symbolic Logic and Mechanical Theorem Proving*, Academic Press Inc., 1973.

[6] Dao, T. T., "Threshold I^2L and Its Application to Binary Symmetric Functions and Multivalued Logic", *IEEE J. Solid-State Circuits*, vol. SC-12, pp.463-472, Oct. 1977.

[7] Dao, T. T., McCluskey E. J. and Russel L. K., "Multivalued Integrated Injection Logic", *IEEE Trans. Comput.*, vol. C-26, pp.1233-1241, Dec. 1977.

[8] Dao, T. T., "Recent Multi-Valued Circuits", *Proc. COMPCON, Spring 81*, pp.194-203, Feb.23-26.

[9] Davio, M. and Deschamps J-P., "Synthesis of Discrete Functions Using I^2L Technology", *IEEE Trans. Comput.*, vol. C-30, pp.653-661, Sept. 1981.

[10] Folberth, O. G., "The Interdependence of Geometrical, Thermal, and Electrical Limitations for VLSI Logic", *IEEE J. Solid-State Circuits*, vol. SC-16, pp.51-52, Feb. 1981.

[11] Forbes, B. E., "Silicon-on-Sapphire Technology Produces High-Speed Single-Chip Processor", *Hewlett-Packard J.*, pp.1-6, Apr. 1977.

[12] Kabat, W. C. and Wojcik A. S., "On the Design of 4-Valued Digital Systems", *IEEE Trans. Comput.*, vol. C-30, pp 666-670, Sept. 1981.

[13] Kabat, W. C. and Wojcik A. S., "Further Results on the Logic Design of Multiple-Valued Circuits Using an Automated Theorem Proving System", *Proc. of The Eleventh Int. Symposium on Multiple-Valued Logic*, pp 135-145, Oklahoma City, Oklahoma, 1981.

[14] Kabat, W.C. and Wojcik A. S., "Automated Synthesis of Combinational Logic Using Theorem Proving Techniques", *Proc. of The Twelfth Int. Symposium on Multiple-Valued Logic*, pp 178-199, Paris, France, 1982.

[15] Kabat, W.C., "Automated Synthesis of Combinational Logic Using Theorem Proving Techniques", PhD dissertation, Illinois Institute of Technology, August 1982.

[16] Kameyama, M. and Higuchi T., "Synthesis of Optimal T-gate Networks in Multiple-valued Logic", *Proc. Ninth Int. Symp. on Multiple-Valued Logic*, pp 190-195, Bath, England 1979.

[17] Keyes, R. W., "The Evolution of Digital Electronics Towards VLSI", *IEEE Trans. Electron Devices*, vol ED-26, pp.271-279, April 1979.

[18] McCharen, J. D., Overbeek R. A. and Wos L., "Problems and Experiments for and with Automated Theorem-proving Programs", *IEEE Trans. Comput.*, vol C-25, pp.773-782, Aug. 1976.

[19] McCluskey, E. J., "Logic Design of Multi-Valued I^2L Logic Circuits", *Proc. Eight Int. Symp. on Multiple-Valued Logic*, pp 14-22, Chicago, Illinois 1978.

[20] McCluskey, E. J., "Logic Design of Multi-Input Quad I^2L Circuits", *Proc. Ninth Int. Symp. on Multiple-Valued Logic*, pp.121-127, Bath, England 1979.

[21] McCluskey, E. J., "Logic Design of Multi-Valued I^2L Logic Circuits", *IEEE Trans. Comput.*, vol C-28, pp.546-559, Aug. 1979.

[22] Moore, G. E., "Are We Really Ready for VLSI", *Proc. Cal-Tech Conf. VLSI*, pp 3-14, Jan. 1979.

[23] Muehldorf, E. I. and Savkar A. D., "LSI Logic Testing - An Overview", *IEEE Trans. Comput.*, vol C-30, pp.1-17, Jan. 1981.

[24] Sedmak, R. M. and Liebergot H. L., "Fault Tolerance of a General Purpose Computer Implemented by Very Large Scale Integration", *IEEE Trans. Comput.*, vol C-29, pp 492-500, June 1980.

[25] Smith, B. T., "User's Guide for the Environmental Theorem Prover, Argonne National Laboratory and Northern Illinois University", Argonne Ntl. Laboratory,

Applied Mathematics Div., Internal Report, 1981.

[26] Smith, K. C., "The Prospects for Multivalued Logic A Technology and Application View", *IEEE Trans. Comput.*, vol. C-30, pp 619-634, Sept. 1981.

[27] Tobias, J. R., "LSI/VLSI Building Blocks", *Computer*, pp 83-101, Aug. 1981.

[28] Winkel, D. and Prosser F., *The Art of Digital Design*, Prentice-Hall Inc., 1980.

[29] Wojciechowski, W. S. and Wojcik A. S., "Multiple-Valued Logic Design by Theorem Proving", *Proc. of The Ninth Int. Symposium on Multiple-Valued Logic*, pp 196-199, Bath, England 1979.

[30] Wojciechowski, W. S., "Multiple-Valued Combinational Logic Design Using Theorem Proving", PhD dissertation, Illinois Institute of Technology, May 1980.

[31] Wos, L., Robinson G. A., Carson D. F. and Shalla L., "The Concept of Demodulation in Theorem Proving", *JACM*, vol. 14, pp.698-709, Oct. 1967.

[32] Wos, L., Robinson J. A. and Carson D., "Efficiency and Completeness of the Set-of-Support Strategy in Theorem Proving", *JACM*, vol. 14, pp 687-697, Oct. 1965.
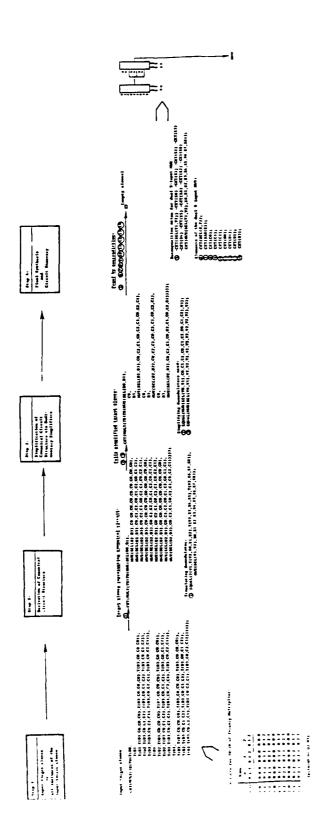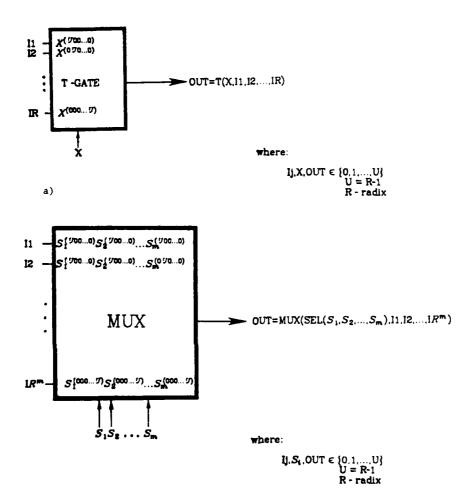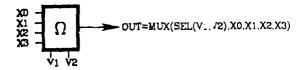
Figure 1 Automated synthesis of logic circuit for TRITO
(least significant trit) of _x2 ternary multiplier.

I1 → $X^{(U00...0)}$
I2 → $X^{(0U0...0)}$

T -GATE

IR → $X^{(000...U)}$

X

OUT=T(X,I1,I2,....,IR)

where:

Ij,X,OUT ∈ {0,1,....,U}
U = R-1
R - radix

a)

I1 → $S_1^{(U00...0)}S_2^{(U00...0)}...S_m^{(U00...0)}$
I2 → $S_1^{(U00...0)}S_2^{(U00...0)}...S_m^{(0U0...0)}$

MUX

IR^m → $S_1^{(000...U)}S_2^{(000...U)}...S_m^{(000...U)}$

$S_1 S_2 ... S_m$

OUT=MUX(SEL($S_1,S_2,....,S_m$),I1,I2,....,IR^m)

where:

Ij,S_i,OUT ∈ {0,1,....,U}
U = R-1
R - radix

b)

Figure 2 Building blocks:
   a) General T-gate block,
   b) General MUX block.

377

$\Omega = (X, V, OUT)$

$X = \{X0, X1, X2, X3\}$ $Xj \in \{0, 1, Vi, -V, RF, OUT\}$
$V = \{V1, V2\}$ $OUT, Vi \in \{0, 1\}$

X0
X1 Ω ──▶ OUT=MUX(SEL(V1, V2), X0, X1, X2, X3)
X2
X3

V1 V2

a)

```
CL -CKT(SEL(V1,V2)) -CKT(X0) -CKT(X1) -CKT(X2) -CKT(X3)
   CKT(MUX(SEL(V1,V2),X0,X1,X2,X3));

   where:  CKT - predicate; denotes that a circuit is constructible,
           MUX - MUX function; denotes MUX logic device,
           SEL - SEL function; denotes selection part of MUX,
           Vi  - control variable,
           Xj  - input variable.
```
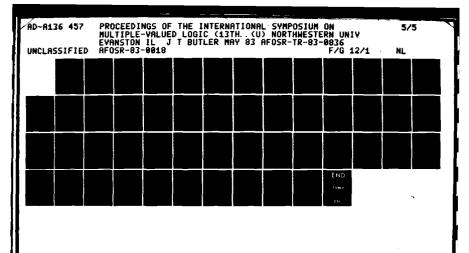
b)

Figure 3 Dual 4-input binary multiplexer: Module example
         (continues, next page):
         a) Formal definition and pictorial representation,
         b) Axiom representation,

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
CL  EQUAL(MUX(SEL(V1,V2),X,X,X,X),X);
CL  EQUAL(MUX(SEL(V1,V2),X,X,X,DX),X);
CL  EQUAL(MUX(SEL(V1,V2),X,X,DX,X),X);
CL  EQUAL(MUX(SEL(V1,V2),X,DX,X,X),X);
CL  EQUAL(MUX(SEL(V1,V2),DX,X,X,X),X);
CL  EQUAL(MUX(SEL(V1,V2),C0,C1,C0,C1),V2);
CL  EQUAL(MUX(SEL(V1,V2),C0,C1,C0,DX),V2);
CL  EQUAL(MUX(SEL(V1,V2),C0,C1,DX,C1),V2);
CL  EQUAL(MUX(SEL(V1,V2),C0,DX,C0,C1),V2);
CL  EQUAL(MUX(SEL(V1,V2),DX,C1,C0,C1),V2);
CL  EQUAL(MUX(SEL(V1,V2),C0,C1,DX,DX),V2);
CL  EQUAL(MUX(SEL(V1,V2),C0,DX,DX,C1),V2);
CL  EQUAL(MUX(SEL(V1,V2),DX,DX,C0,C1),V2);
CL  EQUAL(MUX(SEL(V1,V2),C1,C0,C1,C0),BAR(V2));
CL  EQUAL(MUX(SEL(V1,V2),C1,C0,C1,DX),BAR(V2));
CL  EQUAL(MUX(SEL(V1,V2),C1,C0,DX,C0),BAR(V2));
CL  EQUAL(MUX(SEL(V1,V2),C1,DX,C1,C0),BAR(V2));
CL  EQUAL(MUX(SEL(V1,V2),DX,C0,C1,C0),BAR(V2));
CL  EQUAL(MUX(SEL(V1,V2),C1,C0,DX,DX),BAR(V2));
CL  EQUAL(MUX(SEL(V1,V2),C1,DX,DX,C0),BAR(V2));
CL  EQUAL(MUX(SEL(V1,V2),DX,DX,C1,C0),BAR(V2));
CL  EQUAL(MUX(SEL(V1,V2),C0,C0,C1,C1),V1);
CL  EQUAL(MUX(SEL(V1,V2),C0,C0,C1,DX),V1);
CL  EQUAL(MUX(SEL(V1,V2),C0,C0,DX,C1),V1);
CL  EQUAL(MUX(SEL(V1,V2),C0,DX,C1,C1),V1);
CL  EQUAL(MUX(SEL(V1,V2),DX,C0,C1,C1),V1);
CL  EQUAL(MUX(SEL(V1,V2),C0,C0,DX,DX),V1);
CL  EQUAL(MUX(SEL(V1,V2),DX,DX,C1,C1),V1);
CL  EQUAL(MUX(SEL(V1,V2),C0,DX,C1,DX),V1);
CL  EQUAL(MUX(SEL(V1,V2),DX,C0,DX,C1),V1);
CL  EQUAL(MUX(SEL(V1,V2),C1,C1,C0,C0),BAR(V1));
CL  EQUAL(MUX(SEL(V1,V2),C1,C1,C0,DX),BAR(V1));
CL  EQUAL(MUX(SEL(V1,V2),C1,C1,DX,C0),BAR(V1));
CL  EQUAL(MUX(SEL(V1,V2),C1,DX,C0,C0),BAR(V1));
CL  EQUAL(MUX(SEL(V1,V2),DX,C1,C0,C0),BAR(V1));
CL  EQUAL(MUX(SEL(V1,V2),C1,C1,DX,DX),BAR(V1));
CL  EQUAL(MUX(SEL(V1,V2),DX,DX,C0,C0),BAR(V1));
CL  EQUAL(MUX(SEL(V1,V2),C1,DX,C0,DX),BAR(V1));
CL  EQUAL(MUX(SEL(V1,V2),DX,C1,DX,C0),BAR(V1));
CL  EQUAL(MUX(SEL(V1,V2),C0,C1,C1,C0),PLUS(V1,V2));
CL  EQUAL(MUX(SEL(V1,V2),C0,C1,C1,DX),PLUS(V1,V2));
CL  EQUAL(MUX(SEL(V1,V2),C0,C1,DX,C0),PLUS(V1,V2));
CL  EQUAL(MUX(SEL(V1,V2),C0,DX,C1,C0),PLUS(V1,V2));
CL  EQUAL(MUX(SEL(V1,V2),DX,C1,C1,C0),PLUS(V1,V2));
CL  EQUAL(MUX(SEL(V1,V2),C0,DX,DX,C0),PLUS(V1,V2));
```

c)   where: DX   - don't care function,
                 BAR - multi-valued negation,
                 PLUS - arithmetic plus.

Figure 3 Dual 4-input binary multiplexer: Module example
        (continued): c) Full set of simplifiers.

D1D2D3D4	a	b	c	d	e	f	g
0 0 0 0	1	1	1	1	1	1	0
0 0 0 1	0	1	1	0	0	0	0
0 0 1 0	1	1	0	1	1	0	1
0 0 1 1	1	1	1	1	0	0	1
0 1 0 0	0	1	1	0	0	1	1
0 1 0 1	1	0	1	1	0	1	1
0 1 1 0	0	0	1	1	1	1	1
0 1 1 1	1	1	1	0	0	0	0
1 0 0 0	1	1	1	1	1	1	1
1 0 0 1	1	1	1	0	0	1	1

a)

```
CKT(DISP(MUX(SEL(D4,D2),MUX(SEL(D3,D1),C1,C1,C1,DX),
MUX(SEL(D3,D1),CO,DX,CO,DX),MUX(SEL(D3,D1),CO,C1,C1,DX),MUX(SEL(D3,
D1),C1,DX,C1,DX)),MUX(SEL(D4,D2),MUX(SEL(D3,D1),C1,C1,C1,DX),
MUX(SEL(D3,D1),C1,DX,CO,DX),MUX(SEL(D3,D1),C1,C1,C1,DX),MUX(SEL(D3,
D1),CO,DX,C1,DX)),MUX(SEL(D4,D2),MUX(SEL(D3,D1),C1,C1,CO,DX),
MUX(SEL(D3,D1),C1,DX,C1,DX),MUX(SEL(D3,D1),C1,C1,C1,DX),MUX(SEL(D3,
D1),C1,DX,C1,DX)),MUX(SEL(D4,D2),MUX(SEL(D3,D1),C1,C1,C1,DX),
MUX(SEL(D3,D1),CO,DX,C1,DX),MUX(SEL(D3,D1),CO,CO,C1,DX),MUX(SEL(D3,
D1),C1,DX,CO,DX)),MUX(SEL(D4,D2),MUX(SEL(D3,D1),C1,C1,C1,DX),
MUX(SEL(D3,D1),CO,DX,C1,DX),MUX(SEL(D3,D1),CO,CO,CO,DX),MUX(SEL(D3,
D1),CO,DX,CO,DX)),MUX(SEL(D4,D2),MUX(SEL(D3,D1),C1,C1,CO,DX),
MUX(SEL(D3,D1),C1,DX,C1,DX),MUX(SEL(D3,D1),CO,C1,CO,DX),MUX(SEL(D3,
D1),C1,DX,CO,DX)),MUX(SEL(D4,D2),MUX(SEL(D3,D1),CO,C1,C1,DX),
MUX(SEL(D3,D1),C1,DX,C1,DX),MUX(SEL(D3,D1),CO,C1,C1,DX),MUX(SEL(D3,
D1),C1,DX,CO,DX))));
```

b)

```
CKT(DISP(MUX(SEL(D4,D2),C1,CO,PLUS(D3,D1),C1),MUX(SEL(D4,
D2),C1,BAR(D3),C1,D3),MUX(SEL(D4,D2),BAR(D3),C1,C1,C1),MUX(SEL(D4,
D2),C1,D3,D3,BAR(D3)),MUX(SEL(D4,D2),C1,D3,CO,CO),MUX(SEL(D4,D2),
BAR(D3),C1,D1,BAR(D3)),MUX(SEL(D4,D2),PLUS(D3,D1),C1,PLUS(D3,D1),
BAR(D3))));
```

Demodulators used:

```
EQUAL(MUX(SEL(VO,V1),CO,C1,C1,DX),PLUS(VO,V1));
EQUAL(MUX(SEL(VO,V1),C1,DX,CO,DX),BAR(VO));
EQUAL(MUX(SEL(VO,V1),C1,C1,CO,DX),BAR(VO));
EQUAL(MUX(SEL(VO,V1),CO,DX,C1,DX),VO);
EQUAL(MUX(SEL(VO,V1),CO,CO,C1,DX),VO);
EQUAL(MUX(SEL(VO,V1),CO,C1,CO,DX),V1);
EQUAL(MUX(SEL(VO,V1),V2,DX,V2,DX),V2);
EQUAL(MUX(SEL(VO,V1),V2,V2,V2,DX),V2);
```

c)

Figure 4 Seven-Segment Display binary function:
   a) Truth table,
   b) Target clause representing canonical circuit,
   c) Fully simplified target clause,
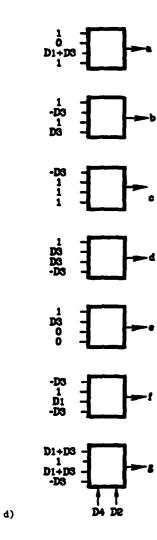
380

d)

Figure 4 Seven-Segment Display binary function:
(continued): d) Logic circuit obtained by
theorem prover.

$\Omega = (X, V, OUT)$

$X = \{X0, X1, X2, X3, X4, X5, X6, X7, X8\}$
$Xj \in \{0, 1, 2, V1, -V1, RF, OUT\}$

$V = \{V1, V2\}$ $OUT, V1 \in \{0, 1, 2\}$

OUT=MUX(SEL(V1,V2),X0,X1,X2,X3,X4,X5,X6,X7,X8)

a)

```
CL   -CKT(SEL(V1,V2)) -CKT(X0) -CKT(X1) -CKT(X2)
     -CKT(X3) -CKT(X4) -CKT(X5) -CKT(X6) -CKT(X7) -CKT(X8)
      CKT(MUX(SEL(V1,V2),X0,X1,X2,X3,X4,X5,X6,X7,X8));
```

b)

```
        EQUAL(MUX(SEL(V1,V2),X,X,X,X,X,X,X,X,X),X);
        EQUAL(MUX(SEL(V1,V2),C0,C0,C0,C1,C1,C1,C2,C2,C2),V1);
        EQUAL(MUX(SEL(V1,V2),C0,C1,C2,C0,C1,C2,C0,C1,C2),V2);
        EQUAL(MUX(SEL(V1,V2),C2,C2,C2,C1,C1,C1,C0,C0,C0),BAR(V1));
        EQUAL(MUX(SEL(V1,V2),C2,C1,C0,C2,C1,C0,C2,C1,C0),BAR(V2));
        EQUAL(MUX(SEL(V1,V2),C0,C1,C2,C1,C2,C0,C2,C0,C1),
              PLUS(V1,V2));
```

c)

Figure 5 Dual 9-input ternary multiplexer: Module example:
        a) Formal definition and pictorial representation,
        b) Axiom representation,
        c) Subset of simplifiers.

DOD1	D3D4 0			1			2		
	0	1	2	0	1	2	0	1	2
0 0	0000	0000	0000	0000	0000	0000	0000	0000	0000
0 1	0000	0001	0002	0010	0011	0012	0020	0021	0022
0 2	0000	0002	0011	0020	0022	0101	0110	0112	0121
1 0	0000	0010	0020	0100	0110	0120	0200	0210	0220
1 1	0000	0011	0022	0110	0121	0202	0220	1001	1012
1 2	0000	0012	0101	0120	0202	0221	1010	1022	1111
2 0	0000	0020	0110	0200	0220	1010	1100	1120	1210
2 1	0000	0021	0112	0210	1001	1022	1120	1211	2002
2 2	0000	0022	0121	0220	1012	1111	1210	2002	2101

a)

```
CKT(MULT(TRIT3(MUX(SEL(D0,D1),MUX(SEL(D2,D3),CO,CO,CO,CO,
CO,CO,CO,CO,CO),MUX(SEL(D2,D3),CO,CO,CO,CO,CO,CO,CO,CO,CO),
MUX(SEL(D2,D3),CO,CO,CO,CO,CO,CO,CO,CO,CO),MUX(SEL(D2,D3),CO,CO,CO,
CO,CO,CO,CO,CO,CO),MUX(SEL(D2,D3),CO,CO,CO,CO,CO,CO,CO,C1,C1),
MUX(SEL(D2,D3),CO,CO,CO,CO,CO,CO,C1,C1,C1),MUX(SEL(D2,D3),CO,CO,CO,
CO,CO,C1,C1,C1,C1),MUX(SEL(D2,D3),CO,CO,CO,CO,C1,C1,C1,C1,C2),
MUX(SEL(D2,D3),CO,CO,CO,CO,C1,C1,C1,C2,C2))),TRIT2(MUX(SEL(D0,D1),
MUX(SEL(D2,D3),CO,CO,CO,CO,CO,CO,CO,CO,CO),MUX(SEL(D2,D3),CO,CO,CO,
CO,CO,CO,CO,CO,CO),MUX(SEL(D2,D3),CO,CO,CO,CO,CO,C1,C1,C1,C1),
MUX(SEL(D2,D3),CO,CO,CO,C1,C1,C1,C2,C2,C2),MUX(SEL(D2,D3),CO,CO,CO,
C1,C1,C2,C2,CO,CO),MUX(SEL(D2,D3),CO,CO,C1,C1,C2,C2,CO,CO,C1),
MUX(SEL(D2,D3),CO,CO,C1,C2,C2,CO,C1,C1,C2),MUX(SEL(D2,D3),CO,CO,C1,
C2,CO,CO,C1,C2,CO),MUX(SEL(D2,D3),CO,CO,C1,C2,CO,C1,C2,CO,C1))),
TRIT1(MUX(SEL(D0,D1),MUX(SEL(D2,D3),CO,CO,CO,CO,CO,CO,CO,CO,CO),
MUX(SEL(D2,D3),CO,CO,CO,C1,C1,C1,C2,C2,C2),MUX(SEL(D2,D3),CO,CO,C1,
C2,C2,CO,C1,C1,C2),MUX(SEL(D2,D3),CO,C1,C2,CO,C1,C2,CO,C1,C2),
MUX(SEL(D2,D3),CO,C1,C2,C1,C2,CO,C2,CO,C1),MUX(SEL(D2,D3),CO,C1,CO,
C2,CO,C2,C1,C2,C1),MUX(SEL(D2,D3),CO,C2,C1,CO,C2,C1,CO,C2,C1),
MUX(SEL(D2,D3),CO,C2,C1,C1,CO,C2,C2,C1,CO),MUX(SEL(D2,D3),CO,C2,C2,
C2,C1,C1,C1,CO,CO))),TRIT0(MUX(SEL(D0,D1),MUX(SEL(D2,D3),CO,CO,CO,
CO,CO,CO,CO,CO,CO),MUX(SEL(D2,D3),CO,C1,C2,CO,C1,C2,CO,C1,C2),
MUX(SEL(D2,D3),CO,C2,C1,CO,C2,C1,CO,C2,C1),MUX(SEL(D2,D3),CO,CO,CO,
CO,CO,CO,CO,CO,CO),MUX(SEL(D2,D3),CO,C1,C2,CO,C1,C2,CO,C1,C2),
MUX(SEL(D2,D3),CO,C2,C1,CO,C2,C1,CO,C2,C1),MUX(SEL(D2,D3),CO,CO,CO,
CO,CO,CO,CO,CO,CO),MUX(SEL(D2,D3),CO,C1,C2,CO,C1,C2,CO,C1,C2),
MUX(SEL(D2,D3),CO,C2,C1,CO,C2,C1,CO,C2,C1)))));
```

b)

```
CKT(MULT(TRIT3(MUX(SEL(D0,D1),CO,CO,CO,CO,MUX(SEL(D2,D3),
CO,CO,CO,CO,CO,CO,CO,C1,C1),MUX(SEL(D2,D3),CO,CO,CO,CO,CO,CO,C1,C1,
C1),MUX(SEL(D2,D3),CO,CO,CO,CO,CO,C1,C1,C1,C1),MUX(SEL(D2,D3),CO,
CO,CO,CO,C1,C1,C1,C1,C2),MUX(SEL(D2,D3),CO,CO,CO,CO,C1,C1,C1,C2,
C2))),TRIT2(MUX(SEL(D0,D1),CO,CO,MUX(SEL(D2,D3),CO,CO,CO,CO,CO,CO,
C1,C1,C1),D2,MUX(SEL(D2,D3),CO,CO,CO,C1,C1,C2,C2,CO,CO),MUX(SEL(D2,
D3),CO,CO,C1,C1,C2,C2,CO,CO,C1),MUX(SEL(D2,D3),CO,CO,C1,C2,C2,CO,
C1,C1,C2),MUX(SEL(D2,D3),CO,CO,C1,C2,CO,CO,C1,C2,CO),MUX(SEL(D2,
D3),CO,CO,C1,C2,CO,C1,C2,CO,C1))),TRIT1(MUX(SEL(D0,D1),CO,D2,
MUX(SEL(D2,D3),CO,CO,C1,C2,C2,CO,C1,C1,C2),D3,PLUS(D2,D3),
MUX(SEL(D2,D3),CO,C1,CO,C2,CO,C2,C1,C2,C1),MUX(SEL(D2,D3),CO,C2,C1,
CO,C2,C1,CO,C2,C1),MUX(SEL(D2,D3),CO,C2,C1,C1,CO,C2,C2,C1,CO),
MUX(SEL(D2,D3),CO,C2,C2,C2,C1,C1,C1,CO,CO))),TRIT0(MUX(SEL(D0,D1),
CO,D3,MUX(SEL(D2,D3),CO,C2,C1,CO,C2,C1,CO,C2,C1),CO,D3,MUX(SEL(D2,
D3),CO,C2,C1,CO,C2,C1,CO,C2,C1),CO,D3,MUX(SEL(D2,D3),CO,C2,C1,CO,
C2,C1,CO,C2,C1)))));
```

Demodulators used:
```
EQUAL(MUX(SEL(V0,V1),CO,C1,C2,C1,C2,CO,C2,CO,C1),PLUS(V0,V1));
EQUAL(MUX(SEL(V0,V1),CO,C1,C2,CO,C1,C2,CO,C1,C2),V1);
EQUAL(MUX(SEL(V0,V1),CO,CO,CO,C1,C1,C1,C2,C2,C2),V0);
EQUAL(MUX(SEL(V0,V1),V2,V2,V2,V2,V2,V2,V2,V2,V2),V2);
```

c)

Figure 6 2x2 Ternary Multiplier function: (continues, next page):
a) Truth table,
b) Target clause representing canonical circuit,
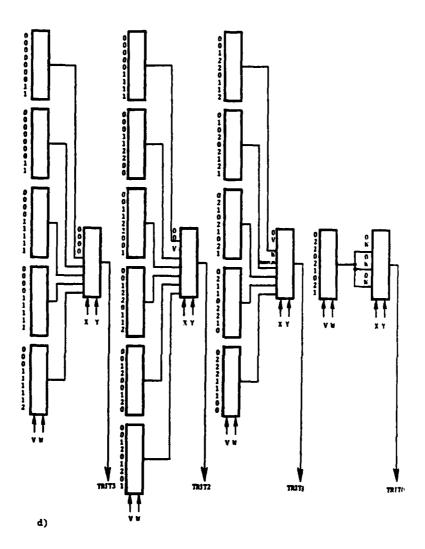c) Fully simplified target clause,

383

d)
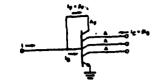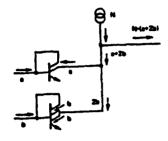
Figure 6 2x2 Ternary Multiplier function: (continued):
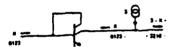d) Logic circuit obtained by theorem prover.

a)

b)

c)

d)

Figure 7 Functions facilitating I^2L circuit design:
a) Signal replication,
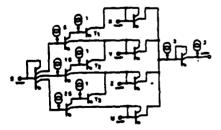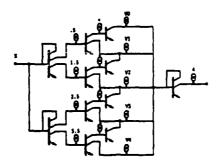b) Weighted sum,
c) Threshold detection,
d) "Complement".

Figure 8 I^2L quaternary T-gate.

385

```
CL  -I2L(X) -I2L(V0) -I2L(V1) -I2L(V2) -I2L(V3) -I2L(V4)
    I2L(T(X,V0,V1,V2,V3,V4));
```

Figure 9 I^{2}L five-valued T-gate and corresponding axiom.

	D1D2					D1D2			
	0	1	2	3		0	1	2	3
D0	0123	0123	0123	0123	D0	0123	0123	0123	0123
0	0123	1230	2301	3012	0	0000	0001	0011	0111
1	1230	2301	3012	0123	1	0001	0011	0111	1111
d	dddd	dddd	dddd	dddd	d	dddd	dddd	dddd	dddd
d	dddd	dddd	dddd	dddd	d	dddd	dddd	dddd	dddd

where: d - don't care condition

a)

```
CL -CKT(ADD(SUM(T(D0,
          T(D1,T(D2,C0,C1,C2),T(D2,C1,C2,C0),T(D2,C2,C0,C1)),
          T(D1,T(D2,C1,C2,C0),T(D2,C2,C0,C1),T(D2,C0,C1,C2)),
          T(D1,T(D2,DX,DX,DX),T(D2,DX,DX,DX),T(D2,DX,DX,DX)))),
          CARRY(T(D0,
          T(D1,T(D2,C0,C0,C0),T(D2,C0,C0,C1),T(D2,C0,C1,C1)),
          T(D1,T(D2,C0,C0,C1),T(D2,C0,C1,C1),T(D2,C1,C1,C1)),
          T(D1,T(D2,DX,DX,DX),T(D2,DX,DX,DX),T(D2,DX,DX,DX)))))));
```

where: CKT - predicate,
       ADD - encapsulating function; Quaternary Full Adder,
       SUM, CARRY - Sum and Carry outputs of the Adder,
       Di - control variable,
       Ci - constant i, (table entry),
       DX - don't care function.

b)

Figure 10 Quaternary Full Adder:
       a) Truth table, b) Input target clause.

```
CL EQU(PLUS(C0,C0),N0);    CL EQU(PLUS(C0,C1),N1);
CL EQU(PLUS(C0,C2),N2);    CL EQU(PLUS(C0,C3),N3);
CL EQU(PLUS(C1,C0),N1);    CL EQU(PLUS(C1,C1),N2);
CL EQU(PLUS(C1,C2),N3);    CL EQU(PLUS(C1,C3),N3);
CL EQU(PLUS(C2,C0),N2);    CL EQU(PLUS(C2,C1),N3);
CL EQU(PLUS(C2,C2),N3);    CL EQU(PLUS(C2,C3),N3);
CL EQU(PLUS(C3,C0),N3);    CL EQU(PLUS(C3,C1),N3);
CL EQU(PLUS(C3,C2),N3);    CL EQU(PLUS(C3,C3),N3);
a)

CL EQU(INH(X,C0),X);
CL EQU(INH(X,C1),C0);
CL EQU(INH(X,C2),C0);
CL EQU(INH(X,C3),C0);
b)

CL EQU(DIFF(C0,X),C0);     CL EQU(DIFF(X,N0),X);
CL EQU(DIFF(C1,N1),C0);    CL EQU(DIFF(C1,N2),C0);
CL EQU(DIFF(C1,N2),C0);    CL EQU(DIFF(C1,N3),C0);
CL EQU(DIFF(C2,N1),C1);    CL EQU(DIFF(C2,N2),C0);
CL EQU(DIFF(C2,N3),C0);    CL EQU(DIFF(C3,N1),C2);
CL EQU(DIFF(C3,N2),C1);    CL EQU(DIFF(C3,N3),C0);
c)
```

Figure 11 Gate specification axioms for quaternary I^2L
connectives:
a) PLUS gate,
b) INHibit gate,
c) DIFFerence gate.


```
CL -EQU(PLUS(X0,Y0),Z0) -EQU(PLUS(X1,Y1),Z1)
   -EQU(PLUS(X2,Y2),Z2) -EQU(PLUS(X3,Y3),Z3)
   I2L(UBB(X0,X1,X2,X3)) I2L(UBB(Y0,Y1,Y2,Y3))
   -I2L(UBB(Z0,Z1,Z2,Z3));

CL -EQU(INH(X0,Y0),Z0) -EQU(INH(X1,Y1),Z1)
   -EQU(INH(X2,Y2),Z2) -EQU(INH(X3,Y3),Z3)
   I2L(UBB(X0,X1,X2,X3)) I2L(UBB(Y0,Y1,Y2,Y3))
   -I2L(UBB(Z0,Z1,Z2,Z3));
a)

CL -EQU(DIFF(X,Y0),Z0) -EQU(DIFF(X,Y1),Z1)
   -EQU(DIFF(X,Y2),Z2) -EQU(DIFF(X,Y3),Z3)
   I2L(UBB(Y0,Y1,Y2,Y3))
   -I2L(UBB(Z0,Z1,Z2,Z3));

b)     where: UBB - Unary Building Block; quaternary vector.
```

Figure 12 Behavioral axioms for quaternary I^2L connectives:
a) PLUS and INHibit gates,
b) DIFFerence gate.
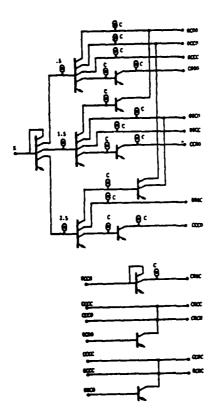
Figure 13 Circuit for generation of quaternary I^2L primitives.

```
CL EQUAL(UBB(CO,X,X,X),NTH(A));
CL EQUAL(UBB(X,CO,X,X),NTH(A));
CL EQUAL(UBB(X,X,CO,X),NTH(A));
CL EQUAL(UBB(X,X,X,CO),NTH(A));
CL EQUAL(UBB(X,CO,CO,CO),NTH(A));
CL EQUAL(UBB(CO,X,CO,CO),NTH(A));
CL EQUAL(UBB(CO,CO,X,CO),NTH(A));
CL EQUAL(UBB(CO,CO,CO,X),NTH(A));
CL EQUAL(UBB(CO,CO,X,X),NTH(A));
CL EQUAL(UBB(X,X,CO,CO),NTH(A));
CL EQUAL(UBB(CO,X,X,CO),NTH(A));
CL EQUAL(UBB(X,CO,CO,X),NTH(A));
CL EQUAL(UBB(CO,X,CO,X),NTH(A));
CL EQUAL(UBB(X,CO,X,CO),NTH(A));
```

Figure 14 Demodulators corresponding to I^2L primitives.
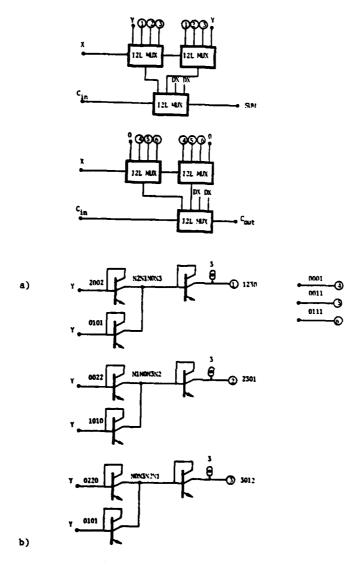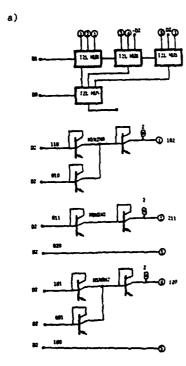
Figure 15 I^2L logic circuit for Quaternary Full Adder:
a) I^2L MUX cascade for Quaternary Full Adder,
b) End-optimized I^2L circuits for composite
unary functions of the Sum and Carry.

389

	D1D2		
	0	1	2
DO	012	012	012
0	102	211	102
1	020	120	210
2	100	012	102

```
-CKT(EX(T(D0,
    T(D1,T(D2,C1,C0,C2),T(D2(C2,C1,C1),T(D2,C1,C0,C2)),
    T(D1,T(D2,C0,C2,C0),T(D2,C1,C2,C0),T(D2,C2,C1,C0)),
    T(D1,T(D2,C1,C0,C0),T(D2,C0,C1,C2),T(D2,C1,C0,C2))));
```

where: EX - example function.

a)
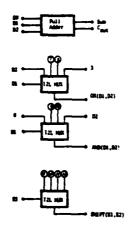
b)
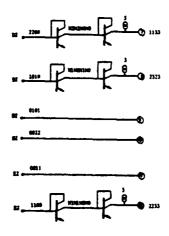
Figure 16 Example of three variable ternary function:
    a) Truth table and input target clause,
    b) I^2L logic circuit.

390

	D1D2			
	0	1	2	3
DO	0123	0123	0123	0123
0	0123	1230	2301	3012
1	1230	2301	3012	0123
d	dddd	dddd	dddd	dddd
d	dddd	dddd	dddd	dddd

a) SUM(D0,D1,D2)

	D1D2			
	0	1	2	3
DO	0123	0123	0123	0123
0	0000	0001	0011	0111
1	0001	0011	0111	1111
d	dddd	dddd	dddd	dddd
d	dddd	dddd	dddd	dddd

CARRY(D0,D1,D2)

	D1			
DO	0	1	2	3
0	0	1	2	3
1	1	1	3	3
2	2	3	2	3
3	3	3	3	3

	D1			
DO	0	1	2	3
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

	E1			
EO	0	1	2	3
0	0	0	1	1
1	2	2	3	3
2	0	0	1	1
3	2	2	3	3

b) OR(D0,D1)   c) AND(D0,D1)   d) SHIFT(E0,E1)

```
CL -CKT(ALU(ADD(SUM(T(D0,
                  T(D1,T(D2,C0,C1,C2,C3),T(D2,C1,C2,C3,C0),
                      T(D2,C2,C3,C0,C1),T(D2,C3,C0,C1,C2)),
                  T(D1,T(D2,C1,C2,C3,C0),T(D2,C2,C3,C0,C1),
                      T(D2,C3,C0,C1,C2),T(D2,C0,C1,C2,C3)),
                  T(D1,T(D2,DX,DX,DX,DX),T(D2,DX,DX,DX,DX),
                      T(D2,DX,DX,DX,DX),T(D2,DX,DX,DX,DX)),
                  T(D1,T(D2,DX,DX,DX,DX),T(D2,DX,DX,DX,DX),
                      T(D2,DX,DX,DX,DX),T(D2,DX,DX,DX,DX)))),
               CARRY(T(D0,
                  T(D1,T(D2,C0,C0,C0,C0),T(D2,C0,C0,C0,C1),
                      T(D2,C0,C0,C1,C1),T(D2,C0,C1,C1,C1)),
                  T(D1,T(D2,C0,C0,C0,C1),T(D2,C0,C0,C1,C1),
                      T(D2,C0,C1,C1,C1),T(D2,C1,C1,C1,C1)),
                  T(D1,T(D2,DX,DX,DX,DX),T(D2,DX,DX,DX,DX),
                      T(D2,DX,DX,DX,DX),T(D2,DX,DX,DX,DX)),
                  T(D1,T(D2,DX,DX,DX,DX),T(D2,DX,DX,DX,DX),
                      T(D2,DX,DX,DX,DX),T(D2,DX,DX,DX,DX)))))),
            OR(T(D0,T(D1,C0,C1,C2,C3),T(D1,C1,C1,C3,C3),
                    T(D1,C2,C3,C2,C3),T(D1,C3,C3,C3,C3))),
           AND(T(D0,T(D1,C0,C0,C0,C0),T(D1,C0,C1,C0,C1),
                    T(D1,C0,C0,C2,C2),T(D1,C0,C1,C2,C3))),
         SHIFT(T(E0,T(E1,C0,C0,C1,C1),T(E1,C2,C2,C3,C3),
                    T(E1,C0,C0,C1,C1),T(E1,C2,C2,C3,C3)))));
```
e)

Figure 17 One Stage Quaternary ALU and its I^2L logic
      circuits (continues, next page):
      a) Quaternary Full ADD operation,
      b) Quaternary OR operator, c) Quaternary AND operator,
      d) Quaternary Left & Right SHIFT operator,
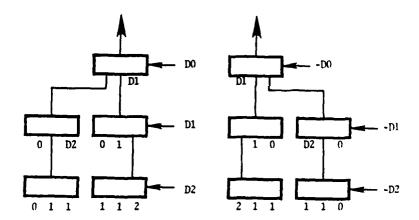   e) Input target clause,

f)

g)

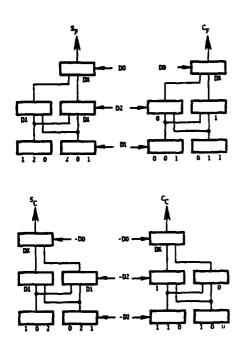Figure 17 One Stage Quaternary ALU and its I^2L logic
circuits (continued):

f) I^2L MUX cascade for One Stage Quaternary ALU,

g) End-optimized I^2L circuits for Sum, Carry,
OR, AND and SHIFT unary functions.

392

DO	D1D2 0 012	1 012	2 012		-DO	-D1-D2 0 012	1 012	2 012
0	000	011	012		0	222	111	000
1	000	111	112		1	211	111	000
2	000	111	222		2	210	110	000

a)   b)

```
CL -CKT(EX(FUNC(T(DO,
            T(D1,T(D2,CO,CO,CO),T(D2,CO,C1,C1),T(D2,CO,C1,C2)),
            T(D1,T(D2,CO,CO,CO),T(D2,C1,C1,C1),T(D2,C1,C1,C2)),
            T(D1,T(D2,CO,CO,CO),T(D2,C1,C1,C1),T(D2,C2,C2,C2)))),
            CMPL(T(DO,
            T(D1,T(D2,CO,CO,CO),T(D2,CO,C1,C1),T(D2,CO,C1,C2)),
            T(D1,T(D2,CO,CO,CO),T(D2,C1,C1,C1),T(D2,C1,C1,C2)),
            T(D1,T(D2,CO,CO,CO),T(D2,C1,C1,C1),T(D2,C2,C2,C2)))))));
```

c)

Figure 18 Sample ternary function:
         a,b) functional and "complementary" truth table,
         c) Input target clause with redundant logic.

```
CL EQUAL(BAR(BAR(X)),X);
CL EQUAL(SLICEC(A1,XP,T(VO,X,Y,Z)),T(BAR(VO),Z,Y,X));
CL EQUAL(SLICEC(A2,XP,T(VO,X,Y,Z)),
        T(BAR(VO),SLICEC(A1,P3,Z),SLICEC(A1,P2,Y),SLICEC(A1,P1,X)));
CL EQUAL(COMPL3(T(VO,X,Y,Z)),
        T(BAR(VO),SLICEC(A2,P3,Z),SLICEC(A2,P2,Y),SLICEC(A2,P1,X)));

LIST GOAL;
CL CKT(ADD(FUNC(X,Y),CMPL(X,Y)));

LIST BACKWARD;
CL -CKT(ADD(FUNC(X,Y),CMPL(X,Y)))
    CKT(ADD(FUNC(X,Y),CMPL(COMPL3(X),COMPL3(Y))));
```

Figure 19 Demodulators for generation of "complementary"
        truth table.

where: EX_F - output of functional logic circuit,

EX_C - output of complementary logic circuit,

$-Di$ - $3-Di$ (negated input variable).

Figure 20 Functional and "complementary" logic circuits for
the sample function in Figure 18.

394

where: S_F , C_F - outputs of functional logic circuits,

S_C , C_C - outputs of complementary logic circuits,

-Di   - 3-Di (negated ternary input variable).

Figure 21 Functional and "complementary" logic circuits for
Ternary Full Adder.

Table 1: Statistics of theorem proving experiments on the automated design.

1. Time requirements [min.sec.tenthsofsec]	
2. Memory requirements [K bytes]	
3. Number of unifications	
4. Clauses Ratio [(clauses kept) / (clauses generated)]	
Primary techniques: hyperresolution and demodulation.	

Function	Statistics
I. Seven-Segment Display Function (module level design)	1. 00.07.18 2. 828 3. 8705 4. 0.869
II. 2×2 Ternary Multiplier (module level design)	1. 00.12.77 2. 1352 3. 15123 4. .839
III. Sample Ternary Function (transistor level design)	1. 00.01.14 2. 792 3. 1226 4. .646
IV. One Stage Quaternary Quaternary ALU (transistor level design)	1. 00.34.16 2. 800 3. 1692 4. .148
V. Ternary Full Adder (internally redundant circuit)	1. 00.19.25 2. 788 3. 192777 4. .925

Comment:

The number of unifications is a measurement which depends upon no particular machine nor a theorem proving system.

Functions references:

I.   In the "2.2 Modular design" section,
II. In the "2.2 Modular Design" section,
III. In the "2.3 Design under Technology Requirements" section,
IV. In the "2.4 Fault Tolerant Logic Design" section,
V.   In the "2.4 Fault Tolerant Logic Design" section.

SYNTHESIS OF MULTIPLE-VALUED LOGIC FUNCTIONS
BASED ON A MODULAR DESIGN APPROACH*

Kwang-Ya Fang

Anthony S. Wojcik

Bell Laboratories
Naperville, Illinois

Illinois Institute of Technolog
Chicago, Illinois

ABSTRACT

As multiple-valued logic technology
advanced rapidly, it is desirable to
develop more cost-effective designs that
effectively make use of modular design
techniques. A decomposition approach to
the modular design of multiple-valued logic
functions was presented previously by the
authors. Based on this approach,
systematic procedures are developed with
simplification techniques which reduce the
number of modules required in the design.
The first step involves the process of
partitioning functions into classes. Then,
the representative building blocks for each
function class are implemented using T-
gates as the basic component. Design
procedures are developed, so that all the
functions in each class can be implemented
directly by the representative building
block.

INTRODUCTION

Most traditional logic design
techniques are based on approaches that
generate designs in a one gate at a time
manner. However, in modern logic design,
the basic units of design are more
complicated than single gates. Certain
logic modules are available as standard
products that are more cost effective than
using discrete gates [1], [2]. The savings
from using standardized components very
often more than offset the inefficiency of
adapting these components to specific
applications. In modern logic design, most
design techniques emphasize modularity.

In multiple-valued logic, however,
very few studies of modular design
techniques have have been presented. A
theoretical study on the synthesis of
multiple-valued logic functions based on
T-gate Universal Logic Modules (ULM's) was

first presented in [3], [4]. A general
approach to the modular design of
multiple-valued logic functions using a
library of modules was presented by the
authors in [5]. In this approach, a
decomposition technique was considered in
terms of available components. The
function is decomposed into subfunctions
which are implemented by a predefined
available set of components. This approach
incorporates the concept of systematically
routing subfunctions to a single output.
The components may consist of a set of
building blocks or a single universal logic
module.

It is desirable to use as few
components as possible from the library.
The techniques for minimizing the number of
components used from a predefined library
was not shown in [5]. In this paper,
techniques are presented to reduce the
number of components required in this
decomposition approach. Examples are given
to illustrate the approach.

THE MODULAR DESIGN APPROACH

The design approach presented in [5]
emphasizes modularity. After a complex
function of an arbitrary number of
variables is decomposed into subfunctions
composed of a small number of variables,
the design of the small subfunctions can be
handled effectively. The entire two-
variable, three-valued function space can
be partitioned into six classes. Based on
these six classes of functions, a set of
building blocks designed with T-gates was
developed, Figure 1. Each structure
implements all the functions in that class,
and in some cases, may require that the
control variables for the T-gates be
functions of one or two variables. In such
cases, the "control functions" are also
implemented by these building blocks.
These building blocks are then used to
implement the original complex function.

The following definitions are given
which will be useful in our discussion.
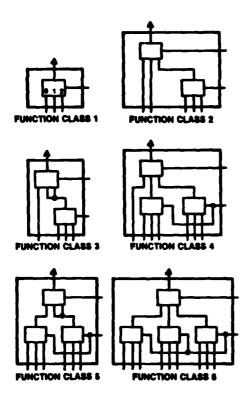
Figure 1. Building Blocks Using T-gates to Implement the Six Classes of Functions

a three-valued, three-variable function with four two-variable subfunctions a, b, c and d. In this partition matrix, column 0, 1, and 2 are trivial columns, and are also unique columns. Column 3 and 4 are identical columns, column 5 and 6, and column 7 and 8 are compatible columns.

x2, x3

x1	00	01	02	10	11	12	20	21	22
0	1	0	a	a	a	b	c	0	2
1	1	1	a	b	b	c	d	2	2
2	1	2	a	c	c	d	b	2	0

Table 1. A 3-valued 3-variable Function

Definition 7: A first level control function is a control function for the first level (or output) T-gate in the building block; and a second level control function is a control function for the second level T-gates in the building block.

Definition 8: If two rows/columns are compatible, the building block input assignment can be identical to the "values" of one row/column, which is called the primary compatible row/column; the other one is called the secondary compatible row/column.

Definition 1: Two rows/columns in a partition matrix are identical iff both have the same "value" at all locations.

Definition 2: Two rows/columns in a partition matrix are compatible iff both contain the same "values".

Definition 3: Two subfunctions are identical or compatible iff the rows/columns in the partition matrix representing these subfunctions are identical or compatible.

Definition 4: A row/column in a partition matrix is a trivial row/column if it is a constant, is equal to a variable, or consists of only one subfunction.

Definition 5: A row/column in a partition matrix is unique iff it is not compatible with any other row/column.

Definition 6: A control function that is implemented by building block X in Figure 1 is called a "structure X" control function.

For example, a control function that is implemented by a building block 2 in Figure 1 is called a "structure 2" control function. The function given in Table 1 is

THE DESIGN OF THREE-VALUED TWO-VARIABLE FUNCTIONS

Because all of the three-valued, two-variable functions are partitioned into six classes, and all the functions in each class can be implemented by the representative building block with or without control functions, the design of a function is easy as long as the function class can be identified and the control function is implemented.

The procedure to determine the class to which an arbitrary three-valued two variable function belongs is the same method used to partition the functions as described in [5]. Essentially, the approach is to group all the functions with the same number of identical/compatible rows/columns into one class. For example, a three-valued two-variable function with two compatible rows/columns in its partition matrix is a function in class 5. Similarly, a function with two compatible rows/columns and a trivial row/column is a function in class 3.

The second part, namely, the identification and implementation of

398

control functions is given in the following procedures.

Procedure 1. Identification of the existence of control functions.

1. A "structure 1" control function exists at the first level if a function in class 2 does not have its non-trivial row/column in the third row/column, or a function in class 3, 4 or 5 does not have its unique row/column in the first row/column.

2. For a function in class 3 or 5 with two compatible rows/columns which are not identical, a structure 2 control function exists at the second level.

3. For a function in class 1 with three compatible rows/columns which are not identical, a structure 4 control function exists.

End Procedure 1.

The design of a "structure 1" control function is trivial. Building blocks 3, 4 and 5 have the unique row/column in the partition matrix implemented at the "0" leg, and building block 2 has its non-trivial row/column in the partition matrix implemented at the "2" leg. For functions in classes 3, 4 or 5 with the unique row/column not at the first row/column, and the functions in class 2 with the non-trivial row/column not at the third row/column, the values of the control variable can be used in such a way so that the unique row/column is routed to the "0" leg.

Procedure 2. The design of "structure 2" and "structure 4" control functions.

1. Identify the control function. A "structure 4" control function only exists in class 1 for functions with three compatible rows/columns, and a "structure 2" control function exists in class 3 or 5 for functions with two compatible rows/columns.

2. Select the primary compatible row/column such that, if numbers are assigned to the rows/columns in the partition matrix in ascending order, the row/column with the smaller number is the primary compatible row/column.

3. Assign values 0, 1 and 2 to the primary compatible and unique row/column from left to right, or top to bottom, in the partition matrix.

4. Assign values 0, 1 and 2 to the secondary compatible rows/columns so that the same value for a primary compatible row/column is assigned to the same subfunction in a secondary compatible row/column.

5. A partition matrix for the control function is defined, and the function can be implemented by building block 2 or 4 with input values assigned as given in the partition matrix.

6. For the "structure 2" control function, if its partition matrix has a non-trivial row/column not in the third row/column, a "structure 1" control function is required as the input to its first level T-gate.

End Procedure 2.

A general procedure is now given for the design of three-valued, two-variable functions.

Procedure 3. The design of three-valued, two-variable functions.

1. Identify the function class to which an arbitrary function belongs.

2. Select the corresponding building block for the function class identified.

3. Subfunction assignment to the inputs of the building block is made.

4. If there exists a "structure 1" control function at the first level for building block 2, 3, 4 or 5, assign the control function output to the first level T-gate control variable.

5. If there exists a structure 2 control function for building block 3 or 5, assign the control function output to the second level T-gate(s) control variable(s).

6. If there exists a structure 4 control function for building block 1, assign the control function output to the T-gate control variable.

End Procedure 3.

These procedures show that the design of a complex function can be performed in a systematic manner. We now discuss techniques to minimize the number of building blocks required in the final design.

VARIABLE SUBSTITUTION

Variable substitution is the process of identifying, and substituting for, function values in a partition matrix with

a variable that specifies the same value. In many cases, variable substitution increases the number of identical or compatible subfunctions, thus resulting in a better design that uses fewer modules. This technique can also be applied directly to the classic decomposition approach [6] in the search for a partition matrix for the function that has the decomposition property under consideration.

Theorem 1: A function value in the partition matrix can be substituted for with a variable iff the variable specifies the same value as the function value to be substituted for.

The procedure that identifies and forms identical subfunctions in a partition matrix is now given.

Procedure 4: Variable substitution to obtain identical subfunctions.

1. Represent the function, $f(X)$, $X = \{x1,x2,...,xn\}$, in a partition matrix where $Xs = \{x1,x2,...,xs\}$ and $Xn-s = \{xs+1,xs+2,...xn\}$.

2. Let Xs be the set of subfunction variables.

3. Let Vji be the function value of subfunction j at the ith position, where $0 <= j <= r^{**}(n-s)$ and $0 <= i <= r^{**}s$, where r is the function radix.

4. Let $Vji(x')$ be the value of variable x' at the ith position, where x' $Xn-s$.

5. Let $i = 0$.

6. If $i > r^{**}s$, subfunctions j and k are identical, stop the process. Otherwise, for two subfunctions j and k, if $Vji = Vki$, increment i and repeat, else goto next step.

7. If $Vji = Vji(x')$ AND $Vki = Vki(x')$, let $Vji = Vki = x'$; increment i and goto previous step; else subfunction j and k cannot be identical to a variable substitution. Stop.

End Procedure 4.

Example 1: Table 2 is a three-valued, three-variable function with three distinct rows. Assume that the function is to be decomposed into two-variable subfunctions, with x2 and x3 as the subfunction variables. By following the steps in Procedure 4, we can substitute for the values 1 and 2 of the first column in the second and third rows with the variable x1. A new partition matrix with two identical rows is shown in Table 3.

x2, x3

x1	00	01	02	10	11	12	20	21	22
0	0	0	0	0	1	0	0	1	0
1	1	1	0	1	1	1	0	1	2
2	2	1	0	1	1	1	0	1	2

Table 2. F(x1,x2,x3) Before Variable Substitution

x2, x3

x1	00	01	02	10	11	12	20	21	22
0	0	0	0	0	1	0	0	1	0
1	x1	1	0	1	1	1	0	1	2
2	x1	1	0	1	1	1	0	1	2

Table 3. F(x1,x2,x3) After Variable Substitution

Using the modular design approach discussed in [5], variables x2 and x3 will be selected as the subfunction variables, and two different subfunctions, g(x2,x3) and h(x2,x3), become the function values of F'(x1). The new unary function F'(x1) and the two subfunctions g(x1,x2) and h(x2,x3) are given in Table 4. Note that in Table 4, subfunction h(x2,x3) has two trivial rows, in which one is a unary function of x3. This can be identified by simply modifying Procedure 4 such that at step 6, instead of comparing subfunctions, we will compare subfunctions with either Xs or Xn-s. Because subfunction h(x2,x3) is a simple two-variable function, the identification of trivial rows or columns can also be done visually. The final design of function F(x2,x2,x3) is shown in Figure 2, which is implemented by using four building blocks as described in Figure 1.

SHARABLE SUBFUNCTION

Sharing of logic devices to implement a function is an important goal when it is desirable to use the fewest number of devices in a design. The logic device can be a complex building block or a basic logic component. We shall first address the issue of the sharing of building blocks to minimize the number of building blocks needed in the design. The sharing of basic logic components will then be discussed.

400

x1

0	1	2
g	h	h

$F'(x1)$

x3

x2	0	1	2
0	0	0	0
1	0	1	0
2	0	1	0

g(x2,x3)

x3

x2	0	1	2
0	x1	1	0
1	1	1	1
2	0	1	2

h(x2,x3)
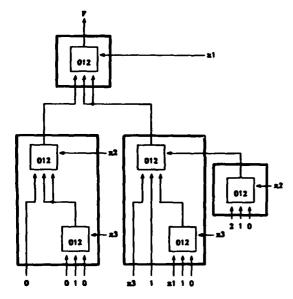
Table 4. The New Unary Function and Two Subfunctions

Figure 2. The Design of Function F(x1,x2,x3)

The following example illustrates the process of identifying and implementing the sharable control function minimization strategy.

Example 2: Two functions, G(x1,x2) and H(x1,x2) are given in Table 5. G(x1,x2) is a member of class 5 with its second row as the unique row, and H(x1,x2) is a member of class 3 with its first row as the unique row. With the substitution of "don't cares" in the unique rows in both G(x1,x2)

and H(x1,x2), the truth table for both functions is shown in Table 6.

x2

x1	0	1	2
0	u	v	w
1	0	1	0
2	w	v	u

G(x1,x2)

x2

x1	0	1	2
0	0	1	2
1	c	b	a
2	a	b	c
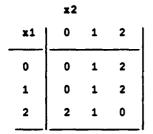
H(x1,x2)

Table 5. Two Functions with Sharable Control Function

The number of distinct rows in the truth table is 3, which is less than or equal to the function radix, so a "structure 2" control function can be shared among G(x1,x2) and H(x1,x2). The design of functions G and H can easily be accomplished with a shared second level structure 2 control function by following Procedure 3 in the previous section. The values 0, 1 and 2 are first assigned to the primary compatible rows and to the unique rows in the truth table, and then the corresponding values are assigned to the secondary compatible rows. The partition matrix for the control function is given in Table 7, and the final design of G(x1,x2) and H(x1,x2) is depicted in Figure 3. Note that in Figure 3, unary functions are assigned in both function H(x1,x2) and the control function shared by functions G and H. Again, these unary functions can be

x1 x2	G	H
0 0	u	-
0 1	v	-
0 2	w	-
1 0	-	c
1 1	-	b
1 2	-	a
2 0	w	a
2 1	v	b
2 2	u	c

Table 6. Truth Table of Functions G(x1,x2) and H(x1,x2)

x2

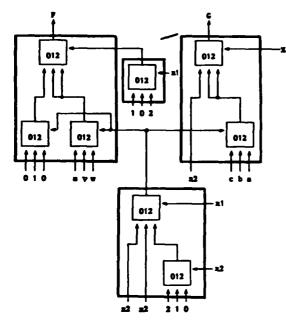| x1	0	1	2
0	0	1	2
1	0	1	2
2	2	1	0

Table 7. The Sharable Control
Function of G and H

Figure 3. The Design of G(x1,x2) and
H(x1,x2) with a Shared
Control Function

identified by Procedure 4 with the simple
modifications in step 6 as mentioned
before; or they can be visually identified
from the partition matrices as shown in
Tables 5 and 7.

Let us now analyze the sharing of
basic logic components without the
restriction of using the building blocks.
The sharing of basic logic components is a
general optimization technique in logic
design, and we shall consider the three-
valued T-gate as the component in the
design. It is interesting to point out
that the T-gate component can be considered
to be a one-variable module; and because it
is functionally complete, it is also a
universal module.

Example 3: The same function $F(x1,x2,x3)$ in
Example 1 is used in this example, and the

three-valued T-gate is considered as the
component to be used in the design. By
permuting the variables, this function can
be described in three different partition
matrices as shown in Table 8. Since the
T-gate can be considered as an one-variable
module, the same modular design approach
can be applied by substituting for the
columns in the matrices with one-variable
subfunctions. The three, two-variable
functions that result after the
substitutions are made are given in Table
9. The five one-variable functions that
result are shown in Table 10.

x2, x3

x1	00	01	02	10	11	12	20	21	22
0	0	0	0	0	1	0	0	1	0
1	1	1	0	1	1	1	0	1	2
2	2	1	0	1	1	1	0	1	2

x1, x3
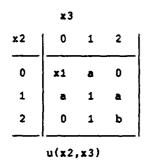
x2	00	01	02	10	11	12	20	21	22
0	0	0	0	1	1	0	2	1	0
1	0	1	0	1	1	1	1	1	1
2	0	1	0	0	1	2	0	1	2

x1, x2

x3	00	01	02	10	11	12	20	21	22
0	0	0	0	1	1	0	2	1	0
1	0	1	1	1	1	1	1	1	1
2	0	0	0	0	1	2	0	1	2

Table 8. Ternary Function F(x1,x2,x3)

Note that subfunctions $d(x2)$ and $e(x3)$
have variable x1 as a function value
indicating that a variable substitution was
performed in the partition matrices in
Table 8 to reduce the number of
subfunctions. In fact, we have reduced the
number of subfunctions from 3 to 2 in
$v(x1,x3)$, and from 3 to 1 in $w(x1,x2)$.
Because $w(x1,x2)$ in Table 9 which is
derived from the last partition matrix in
Table 8, requires the minimum number of
subfunctions, (only one subfunction $e(x3)$
is needed), we select the new two-variable
function $w(x1,x2)$. Similarly, we select x2

x3

x2	0	1	2
0	x1	a	0
1	a	1	a
2	0	1	b

u(x2,x3)

x3

x1	0	1	2
0	0	c	0
1	d	1	x2
2	d	1	x2

v(x1,x3)

x2

x1	0	1	2
0	0	e	e
1	e	1	x3
2	e	1	x3

w(x1,x2)

Table 9. The New Two-variable Functions of F(x1,x2,x3)

x1

0	1	2
0	1	1

a(x1)

x1

0	1	2
0	2	2

b(x1)

x2

0	1	2
0	1	1

c(x2)

x2

0	1	2
x1	1	0

d(x2)

x3

0	1	2
x1	1	0

e(x3)

Table 10. Five Unary Functions Substituted in Table 9

as the variable for subfunction from w(x1,x2). The final design is shown in Figure 4.

Comparing Figures 2 and 4, one observes that the solution in Figure 2 requires two more T-gates then Figure 4. However, Figure 2 indicates that F(x1,x2,x3) is implemented with four building blocks from the components library defined in Figure 1, while Figure 4 indicates that F(x1,x2,x3) is implemented by four T-gates which can be considered to
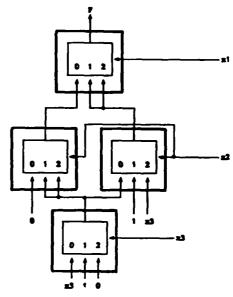
Figure 4. Optimal Logic Circuit for F(x1,x2,x3)

be four one-variable building blocks. Although the design goals are different in Figures 2 and 4, the general modular design approach presented in [5] is independent of the components selected and can be directly applied with respect to the logic components to be used in the design. It is interesting to point out that the example we have chosen, F(x1,x2,x3), is taken from [7]. The design method presented by Kabat and Wojcik is an automated technique based entirely on a theorem proving approach to design. The final design of this three-variable function in Figure 4 is identical to the design in [7], an optimal logic circuit for F(x1,x2,x3).

We have considered the sharing of two-variable building blocks and the sharing of basic logic components. It is important to note that the element to be shared is determined by the components used in the design. For example, the sharing of T-gates is not applicable in the first example because the design is restricted to a predefined component library. That is, the component within a building block cannot be shared among building blocks. However, if we consider the design by using the T-gate as the building block, then the sharing of a T-gate can be considered, as shown in the second example.

SELECTION OF CONTROL VARIABLES

Another important consideration in reducing the number of modules needed in a logic design is the selection of control variables at each level of the tree

403

structure of the design. In [3], this problem was discussed and defined as "the proper order of the expansion", which is a technique of finding the most imcomplete tree structure. The modular design approach in [5] has a similar characteristic, namely, to decompose the function into subfunctions such that the fewest number of subfunctions is selected. In this section, we will discuss a direct application of this technique at the lowest level, namely, the implementation of the subfunctions.

If we have decided to decompose the function into two-variable subfunctions which will then be implemented by building blocks from a components library as shown in Figure 1, the selection of which variable to be used as the first level control variable results in different building blocks being selected. For example, consider a three-valued, two-variable function $A(x_1,x_2)$ which has subfunction a, b, c and d as its function values as shown in Table 11. This function can be considered to be a function in class 5, that is, there exist two compatible rows. It can also be considered as a function in class 6, namely, there is no trivial or compatible columns. However, it is obvious that the selection of a building block 1 is a better choice than a building block 6. Not only is building block 5 a simpler component, but also the interconnections with the lower level subfunctions, a, b, c and d are much more simplified than if a building block 6 is selected. Figures 5 and 6 show the difference of the designs based on building blocks 5 and 6, respectively.

Figure 5. The Design of $A(x_1,x_2)$ with x_1 as the First Level Control Variable

x2

x1	0	1	2
0	c	d	a
1	a	b	c
2	a	b	c

Table 11. Subfunction $A(x_1,x_2)$
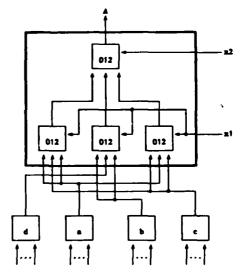
Figure 6. The Design of $A(x_1,x_2)$ with x_2 as the First Level Control Variable

It is clear that the design in Figure 5 is a better solution. Therefore, for each two-variable function to be implemented by a building block from the library, the variable with the largest number of trivial or identical subfunctions should be selected as the first level control variable.
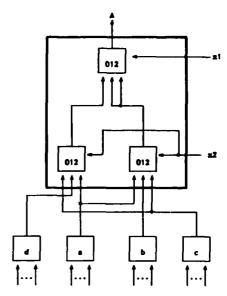
COMPREHENSIVE EXAMPLE

Let us now consider a five-variable function as an example to illustrate the general design approach in [5] and the optimization techniques discussed in this paper. This example, taken from [8], is a function which can be decomposed into a form called a multiple complex disjunctive decomposition.

404

Table 12 shows a three-valued, five-variable function. The building blocks in Figure 1 are assumed to be the components used to implement the function. As shown in [5], this function can be decomposed into two-variable subfunctions recursively until the function is expressed as a one-variable function k(x5), Table 13, which incorporates two subfunctions g(x3,x4) and h(x3,x4). The subfunctions g and h are shown in Table 14. These, in turn, consist of five subfunctions a, b, c, d and e of variables x1 and x2, as shown in Table 15.

In this example, k(x5) is a single variable function, and it is obvious that it can be easily implemented by building block 1. Because both subfunctions g and h have two compatible columns with the unique column as the first column, they can be implemented by using two building block 5s with a second level structure 2 control function. With the substitution of a "don't care" in the unique rows in both g and h, the truth table of both subfunctions is shown in Table 16. The number of distinct rows in the truth table is 2, so a structure 2 control function which is defined in Table 17, can be shared between g and h.

The lowest level design is the implementation of the subfunctions a, b, c, d and e. Again, since all these five subfunctions have three compatible columns, they can be implemented by using five building block 1s with a "structure 4" control function. Similarly, the number of distinct rows of their truth tables is 3, so a "structure 4" control function can be shared among all five subfunctions. The truth table for all five subfunctions is given in Table 18. The partition matrix for the control function is given in Table 19. The final design of this five-variable function is shown in Figure 7, which shows all three levels of subfunctions with their shared control functions.

	x5		
	0	1	2
	g	h	g

k(x5)

Table 13. Final Representation of f(x1,x2,x3,x4,x5)

g(x3,x4)

x4 \ x3	0	1	2
0	a	b	b
1	b	b	a
2	c	a	b

h(x3,x4)

x4 \ x3	0	1	2
0	0	d	d
1	d	d	0
2	e	0	d

Table 14. Subfunctions of F(x3,x4,x5)

x5,x4,x3

```
           0 0 0 0 0 0 0 0 0   1 1 1 1 1 1 1 1 1   2 2 2 2 2 2 2 2 2
           0 0 0 1 1 1 2 2 2   0 0 0 1 1 1 2 2 2   0 0 0 1 1 1 2 2 2
  x2,x1    0 1 2 0 1 2 0 1 2   0 1 2 0 1 2 0 1 2   0 1 2 0 1 2 0 1 2
  -----------------------------------------------------------------------
    00      1 0 0 0 0 1 1 1 0   0 1 1 1 1 0 2 0 1   1 0 0 0 0 1 1 1 0
    01      1 2 2 2 2 1 0 1 2   0 0 0 0 0 0 2 0 0   1 2 2 2 2 1 0 1 2
    02      2 0 0 0 0 2 1 2 0   0 2 2 2 2 0 0 0 2   2 0 0 0 0 2 1 2 0

    10      2 0 0 0 0 2 1 2 0   0 2 2 2 2 0 0 0 2   2 0 0 0 0 2 1 2 0
    11      1 0 0 0 0 1 1 1 0   0 1 1 1 1 0 2 0 1   1 0 0 0 0 1 1 1 0
    12      1 2 2 2 2 1 0 1 2   0 0 0 0 0 0 2 0 0   1 2 2 2 2 1 0 1 2

    20      1 2 2 2 2 1 0 1 2   0 0 0 0 0 0 2 0 0   1 2 2 2 2 1 0 1 2
    21      1 0 0 0 0 1 1 1 0   0 1 1 1 1 0 2 0 1   1 0 0 0 0 1 1 1 0
    22      2 0 0 0 0 2 1 2 0   0 2 2 2 2 0 0 0 2   2 0 0 0 0 2 1 2 0
```

Table 12. f(x1,x2,x3,x4,x5)

a(x1,x2)

x1 \ x2	0	1	2
0	1	2	1
1	1	1	1
2	2	1	2

b(x1,x2)

x1 \ x2	0	1	2
0	0	0	2
1	2	0	0
2	0	2	0

c(x1,x2)

x1 \ x2	0	1	2
0	1	1	0
1	0	1	1
2	1	0	1

d(x1,x2)

x1 \ x2	0	1	2
0	1	2	0
1	0	1	1
2	2	0	2

e(x1,x2)

x1 \ x2	0	1	2
0	2	0	2
1	2	2	2
2	0	2	0

Table 15. Subfunctions of f(x1,x2,x3,x4,x5)

Table 17. (x4 columns, x3 rows)

x3 \ x4	0	1	2
0	0	1	2
1	0	1	2
2	0	2	1

Table 17. The Sharable Control Function of g and h

x2	x1	a	b	c	d	e
0	0	1	0	1	1	2
0	1	1	2	0	0	2
0	2	2	0	1	2	0
1	0	2	0	1	2	0
1	1	1	0	1	1	2
1	2	1	2	0	0	2
2	0	1	2	0	0	2
2	1	1	0	1	1	2
2	2	2	0	1	2	0

Table 18. Truth Table of Subfunction a, b, c, d and e

x3	x4	g	h
0	0	-	-
0	1	-	-
0	2	-	-
1	0	b	d
1	1	b	d
1	2	a	0
2	0	b	d
2	1	a	0
2	2	b	d

Table 16. Truth Table of Subfunctions g(x1,x2) and h(x1,x2)

Table 19. (x1 columns, x2 rows)

x2 \ x1	0	1	2
0	0	1	2
1	2	0	1
2	1	0	2

Table 19. Control Function for a, b, c, d and e

406
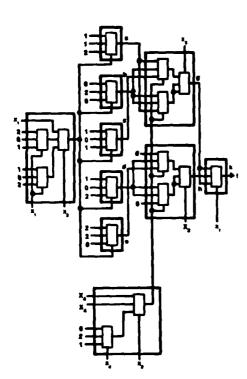
Figure 7. T-gate Building Block Solution for Function f(x1,x2,x3,x4,x5)

CONCLUSION

In this paper, systematic techniques to reduce the number of components required to implement a function, based on the approach in [5], were presented. These techniques are, in principle, applicable to functions in any radix. The goal is to minimize the number of building blocks used from a predefined component library, but not to minimize the number of basic logic components from which the building blocks are composed. However, this approach does not always guarantee an optimal solution, namely, the least number of building blocks. It should be noted that the optimal design is a function of the cost criterion used, such as the number of components or modules used, the complexity of interconnections between components or modules, etc. Further studies are needed in the design of optimal circuits with respect to selected cost criterion.

References

1. T. R. Blakeslee, *Digital Design with Standard MSI and LSI*, John Wiley and Sons, 1979.

2. W. D. Becher, *Logical Design Using Integrated Circuits*, Hayden Book Co., 1977.

3. T. Higuchi and M. Kameyama, "Synthesis of Multiple-Valued Logic Networks Based on Tree-Type Universal Logic Modules", *IEEE Trans. Comput.*, vol. C-26, pp. 1297-1302, December 1977.

4. M. Kameyama and T. Higuchi, "Synthesis of Optimal T-gate Networks in Multiple-Valued Logic", *Proc. of the 1979 International Symposium on Multiple-Valued Logic*, pp. 190-195, May 1979.

5. K. Y. Fang and A. S. Wojcik, "An Approach to The Modular Design of Multiple-Valued Logic Functions", *Proc. of the 1982 International Symposium on Multiple-Valued Logic*, pp. 260-266, May 1982.

6. H. A. Curtis, *A New Approach to The Design of Switching Circuits*, Van Nostrand, 1962.

7. W. C. Kabat and A. S. Wojcik, "Automated Synthesis of Combinational Logic Using Theorem Proving Techniques", *Proc. 1982 International Symposium on Multiple-Valued Logic*, pp. 178-199, May 1982.

8. S. Thelliez, *Introduction to the Study of Ternary Switching Structures*, Gordon and Breach Science Publishers, 1973.

SYNTHESIS ALGORITHM FOR MINIMAL COMPONENTS IN T-ULM NETWORKS

Powsiri Klinkhachorn & Robert Swartwout
Department of Electrical Engineering
West Virginia University
Morgantown, W.V., U.S.A. 26506

Abstract

An algorithm has been developed and programmed in Pascal to design T-ULM networks for MVL systems with up to 6 r-valued inputs (r from 2 to 5). The algorithm is based on a special form of T-ULM formed from threshold detectors and switches. The objective of the algorithm is to minimize the total number of components (thresholds and switches) and also the total number of T-ULM's. Techniques are presented that make optimal use of don't care conditions in the functional specification.

1.0 Introduction

It appears that there is a need to develop a design technique for multi-valued logic systems which is simple and yields practical though not necessarily absolutely minimal circuits. Several authors have indicated that such a design technique has been developed, based on Universal Iterative tree structures or the Tree-Type universal logicl module (T-ULM's) [6,9,17,21]. This design approach can be applied to all switching functions, both combinational and sequential. Singh [17] described a multi-valued logic design technique using T-ULM's in a tree structure. However, there was no attempt to optimize the network. Higuchi [6,7,8], Kameyama [11,12] and Wojciechowski [21] have each used T-ULM's in a tree structure and have showed synthesis methods to minimize the number of T-gates. However, it is the intent of this research effort to develop a synthesis method that will minimize the component count within a specific form of T-ULM as well as the number of T-gates. This method will also allow the use of don't care states in the functional specification.

1.1 Statement of the Problem

The overall objective of this research effort was to develop a synthesis procedure to optimize a multi-valued logic network when using a new T-gate circuit design. The research effort was subject to the following conditions:

1. The synthesis procedure, based on the proposed circuit, was to be algorithmic.
2. The algorithm should be programmed in Pascal so as to execute in a reasonable amount of time.
3. The program should accept up to 6 input variables and generate solutions for a radix value (ranging from 2 through 5).
4. *Don't care conditions were acceptable as functional values for F(X).*
5. The function was to be specified in the form of a table or a map.
6. The solution would be considered optimal in terms of the total component count (sum of thresholds and switches).

2.0 Historical Summary

During the past decade, several algebraic approaches to form a theory of r-valued switching systems have appeared. These r-valued algebras are the basis of multi-valued logic (MVL) switching theory. An early paper by Allen and Givone [1] used binary valued literal gates and the use of thresholds. Another by Vranenic [20] introduced cyclic gates and the use of analog techniques in MVL implementations. Very recently, McCluskey [14] introduced a method for designing multi-input, MVL IIL circuits. Pomper [16] showed an efficient method for finding the representation of an MVL function which is applicable to both traditional *min/max operators*, as well as the IIL *sum/product* operators.

Two types of MVL elements based on multi-threshold gates have been investigated. The multi-threshold radix-R MT(R) gate [5] was described by Druzeta and shows potential for use in some applications. Ishizuka [9,10] developed a synthesis method for multi-valued multi-threshold networks using IIL circuits. Applications of multi-threshold logic have also been shown by Current [3,4].

The simpler concept in designing MVL networks based on the use of T-gates was shown to be attractive and practical [7]. Higuchi and Kameyama [6,7,11,12] developed a synthesis method for MVL networks based on tree-type universal logic modules or T-ULM's. Singh [17] showed the use of universal iterative tree structures in designing combinational and sequential MVL circuits. Wojciechowski [21] and Kabat [13] used a Theorem Prover technique to design T-gate networks.

2.1 Mathematical Properties

Consider an r-valued system where all elements are members of the set $R = [0,1,2,...,r-1]$. Let there be n input variables, all members of the input set $X = [x1,x2,...,xn]$. Then a function of the inputs would be $F(x1,x2,..,xn)$ where $f,xi \in R$. **Definition 1.** An individual T-gate with one control input is defined in Equation 1 and is shown in Figure 1. This gate would realize the function $f(s)$.
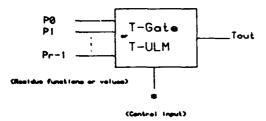
Figure 1. An r-valued T-gate or T-ULM symbol

(Residue functions or values)

(Control Input)

Figure 2. Universal Logic Module structure

$$f(s) = Tout = T(P;s) = T(p0,p1,...pr-1:s) \qquad (1)$$
where $P = [p0,p1,...,pr-1]$
where $f(s), pj, s \in R$.
and where $Tout = pi$ when $s=i$

Singh has shown [17] that a function of n input variables can be realized using a tree structure as shown in Figure 2.

Theorem 1. Let Xc be a single input variable from the input set X, then define X', $Xc \cap X' = 0$, the null set, and $Xc \cup X' = X$. An arbitrary function of n variables can be expressed in the following form [17]:

$$F(X) = T(P(X');Xc) = T(pj(X');Xc) \qquad (2)$$
where each $pj(X') = F(X',Xc=j)$.

Kameyama [6] showed that for any arbitrary logic function of n variables, the maximum number of T-gates required is $(r^{**}n-1)/(r-1)$.

The T-gate of Figure 1 can be made to perform almost any operation by proper selection of the pi values. As a result, these units are usually referred to as T-gate universal logic modules or T-ULM's.

Definition 2. A generalized form of the T-ULM shown in Figure 1 is one in which the output is determined by k control variables of the control vector $C = (c1,c2,...ck)$. The number of unique states for the vector C is $r^{**}k$. Let the scalar S be the one to one mapping of C, that is, $C \longleftrightarrow S = \sum c_i r^{i}(0 < S < (r^{**}k))$. Also let $P = [p0,p1,..pr-1]$. P is called the vector of residue functions. The output of the T-ULM is then
$$Uout = U(P;C) = U(p0,p1,...,pr^{**}k-1;C)$$
$$Uout = pj \text{ when } C \longleftrightarrow S = j.$$

Axiom 1. If all elements of P are the same, i.e. $p0=p1=...=pj=...=p$, then $U(P;C)=p$ for all values of C.

Theorem 2. Consider an extension of Theorem 1. Let the vector X be the set of n variables (x1, x2,...,xn), and Xc, a subvector of X, where $X' \cap Xc = 0$ and $X' \cup Xc = X$. Then
$$U(p0(X),..,pj(X),...,pr^{**}k-1;Xc) =$$
$$U(p0(X',Xc \longleftrightarrow 0),..,pj(X',Xc \longleftrightarrow j),..,$$
$$pr^{**}k-1(X',Xc \longleftrightarrow r^{**}k-1);Xc).$$

3.0 Synthesis of MVL Functions with T-Gate Logic

Theorem 2 illustrates how many logic functions of several variables can be realized with T-ULM's in a tree structure. However, as the number of variables of a function becomes large, the number of T-ULM's necessary to realize the function increases exponentially in a canonical expression. Higuchi and Kameyama [6,11,12] proposed algorithms to synthesize a network of T-ULM's through (1) the use of functional decomposition and (2) the identification of trivial residue functions. Their
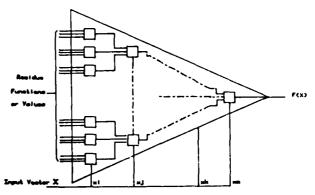
objective was to reduce the number of T-ULM's in the logic networks. The first method is very effective and yields a minimum solution. But Butler [2] showed that only a small percentage of all functions can be realized using this method. The second method attempts to minimize the function by using the elimination of trivial T-ULM's. To minimize the number of T-gates, using the property of trivial functions, the ordering of the input variables in each level of the tree has to be evaluated. Figure 3 shows the functional table and the tree structure of a T-gate network which is to be minimized by eliminating trivial functions. As will be shown later, the algorithm presented here will facilitate additional minimization in the number of T-gates and the components used to produce those T-gates.

3.1 Reduced Component Count in T-Gate Logic

Definition 3. Let the threshold detector and switch shown in Figure 4 be the primitive components in a T-gate circuit. The threshold detector and switch have the following characteristics:

Threshold detector:
  If input > reference, then output:=True
                        else output:=False
  Switch: If control=True, then output:=input
                        else output:=floating state

Any 1-input T-gate can be built from the primitive components as defined in Definition 3. A base-4 T-gate circuit diagram is shown in Figure 5.

Theorem 3. For any r-valued logic function, the maximum number of thresholds and switches required
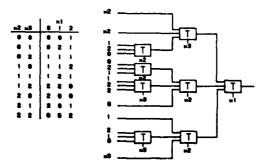

Figure 3. Functional table and tree structure network (Example of Kameyama and Higuchi)

409

to construct a T-gate circuit are (r-1) and 2(r-1) respectively.

Proof: Since there are r discrete levels of signal to be differentiated, it is obvious that there are (r-1) gaps between these levels. Therefore, the maximum number of thresholds required is equal to (r-1). The number of switches needed is twice that number, that is 2(r-1).

Theorem 4. An arbitrary function of n variables, each r-valued, can be completely realized by a T-ULM network with a maximum of $(r^{**}n)-1$ thresholds and $2(r^{**}n)-1$ switches.

Proof: Higuchi [6] showed that a maximum of $(r^{**}n)-1/(r-1)$ T-ULM's are needed to form any arbitrary function of n variables. From Theorem 3, each T-gate requires (r-1) thresholds and 2(r-1) switches. Therefore, a maximum of $(r^{**}n)-1$ thresholds and $2((r^{**}n)-1)$ switches are required to express an n variable, r-valued function.

Axiom 2. Any 1-input r-valued function can be realized from a single T-ULM with a maximum of (r-1) thresholds and 2(r-1) switches.

Axiom 3. A 1-input r-valued function can be completely eliminated if the value of the function is equal to a constant.

Axiom 4. A 1-input r-valued function can be replaced by the input variable if the value of the residues corresponds to the distinct radix values of the variable.

Theorem 5. The number of thresholds and switches used in a 1-input r-valued T-gate can be reduced by one and two respectively if two adjacent values of the function are the same. Obviously, if two adjacent values of the function are the same, then there is no need to detect another level. Therefore, the number of thresholds and switches can be reduced by 1 and 2 respectively.

3.2 Don't Care Assignment

If one considers an incompletely specified function, it is obvious that Theorem 5 and Axioms 3 and 4 can be applied to reduce the number of thresholds and switches.

Axiom 5. If the value of a don't care term is set equal to the value of one of the adjacent terms of the 1-input r-valued function, the number of thresholds and switches required is reduced. Each such don't care assignment will save one threshold and two switches.

Axiom 6. Whenever possible, the value of a don't care term should be assigned a value that will cause the 1-input function to be a trivial func-
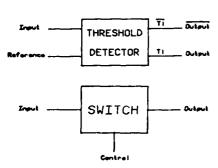

Figure 4. Block diagram of Threshold Detector and Switch

tion. This type of assignment will save at least one T-gate.

Theorem 6. If every residue value of a 1-input r-valued function is a don't care, then using Axioms 3 or 4, the T-gate can be eliminated. If the T-gate tree structure has more than one level, one threshold element and 2 switches in the logic level beyond this one are also saved.

Proof: If one selects all don't care values equal to the same constant or equal to the appropriate value of the input variable, then a T-gate can be completely eliminated. However, if this 1-input r-valued function is a subtree of a T-gate network and one assumes that the value of the don't care terms are equal to the value of an adjacent sub-tree, then from Theorem 5 it can be shown that not only this 1-input r-value function can be eliminated, but also it can save one more threshold and 2 switches for a T-gate in the next level of this subtree. An example of assigning don't care terms is shown in Figure 6.

3.3 Synthesis Procedure

The method used in this synthesis of T-gate networks is based on the canonical expansion of the function, which is a generalization of Shannon's expansion theorem for Boolean functions.

Definition 4. Let F(x) be an n-variable r-valued function and let the set of n variables be divided into two disjoint subsets X' (n-1 variables) and x_i (1 variable).

Applying definition 4 to an n-variable r-valued function, construct an n-level tree network from the subtrees or residue functions. Each subtree represents the subfunction of F(X). Each subtree can be exhaustively constructed through repeated application of definition 4 until X' contains only
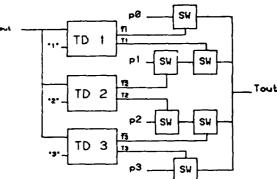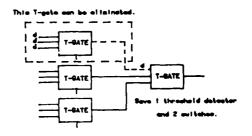

Figure 5. General T-gate circuit

Figure 6. Don't care assignment (Theorem 6)

410

1-variable. If the output level of the T-gate network is called the first level, then each subtree will generate level, 2,3,..,n. The first level will contain r subtrees and each subtree will have n-1 levels. Figure 7 shows the example of decomposing a 4-variable, 3-valued function into subtrees of the T-gate network.

Theorem 7. An n-variable r-valued function requires at least $(r^{**}(n-1))n!$ iterative loops to be checked in order to determine the minimum solution in the T-gate network.

Proof: Since there are n-levels in the T-gate network, then at each level all T-gates have r-values and therefore $r^{**}(n-1)$ loops must be tested. However, for an n-variable function, there are n! possible orders of variables in the n-levels of the tree. As a result, a total of $(r^{**}(n-1))n!$ tests are required.
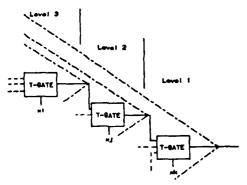
A desirable design is one in which a minimum *number of components are used. This implies a* minimum number of components within the T-gates as well as fewest gates. It is the opinion of the authors that a reduction in the number of thresholds and switches required will also simplify the interconnect problems on a chip. In fabricating the chip, all T-gates may be fabricated identically, but each will have its own unique connection pattern.

3.4 Synthesis Algorithm

The following algorithm accomplishes the explicit enumeration technique that is used to obtain the solution for a function to be realized with a T-gate network.

Algorithm 1. Synthesize a Fan-out Free Network for $F(X)=F(x1,..,xn)$.

1. Select any xi to be the control variable for the first level of the tree in the T-gate network.

2. For each residue value of the given xi, construct a subtree where xj (j=1,2,....,n;j<>1) is the control variable of the next level.

3. Repeat step 2 exhaustively until the nth-level is reached, i.e. X' contains only one variable. Each xi is to be used only once for each subtree.

4. Assign any don't care terms by applying Theorem 6 and axioms 5 and 6 to each 1-input function of the subtrees at level n.

5. Examine all subfunctions to determine if their value will cause further minimization at the next level. Examples of this technique are shown in Figures 6 and 8.

6. Evaluate the total number of thresholds and switches used in level n of all the subtrees.

7. Save the total number of switches and thresholds used for all levels in this particular assignment of the control variables in the tree, including the ordering of the control variable.

8. Permute the control variable for each subtree and repeat steps 3, 4, 5, and 6.

9. Compare the number of thresholds and switches used by each permutation of the control variable of the subtrees. Select the order for each subtree that yields a minimum number of components used.

10. Repeat steps 1 to 9 with another xi.

11. Select the best solution after all of the xi have been exhaustively tested.
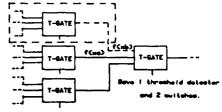
3.5 T-Gate Network Examples

To illustrate the synthesis method of Algorithm 1, we present the design of two T-gate networks:

Example 1.

Consider a 3-valued 2-variable function described by the map below. The numbered steps refer to the steps in Algorithm 1.

X1\X2	0	1	2
0	D	2	1
1	D	0	1
2	D	0	2

1. Select X1 to be the control variable of the first level of the tree in the T-gate network.

2. Since there are only 2-variables, X2 becomes the control variable of the second level. The subtrees of X1 are constructed as follows:

   a) $p0(X2) = T(D, 2, 1; X1=0)$,
   b) $p1(X2) = T(D, 0, 1; X1=1)$, and
   c) $p2(X2) = T(D, 0, 2; X1=2)$.

3. Go to step 4, since X' contains only one level, i.e. X2.

4. Since each subtree contains a don't care term, Theorem 6 and Axioms 5 and 6 can be applied to the functions of a, b, and c shown in step 3. The results are:

   d) $p0(X2) = T(2, 2, 1; X1=0)$,
   e) $p1(X2) = T(0, 0, 1; X1=1)$, and
   f) $p2(X2) = T(0, 0, 2; X1=2)$.

5. The thresholds required in fabricating the $p0(X2)$, $p1(X2)$, and $p2(X2)$ are 1, 1, 1 respectively.

6. Since each subtree is different in value, there can be no further minimization of the first level. This design requires 2 thresholds for the T-gate in level 1.

Figure 7. Subtrees of the T-gate network.

Figure 8. Elimination of subtrees by Theorem 6.

411

7. The total number of thresholds used for this assignment (X1 = first level, and X2 = second level) is 5 units.

8. Since X2 is the only subtree of X1, skip to step 10.

9. Skipped.

10. Repeat steps 1-9 with X2 as the first level. By following the same process as shown above, it can be shown that

a) $p0(X1) = T(D, D, D; X2=0)$,

b) $p1(X1) = T(2, 0, 0; X2=1)$, and

c) $p2(X1) = T(1, 1, 2; X2=2)$.

After assigning values to the don't care terms,

d) $p0(X1) = T(2, 0, 0; X2=0)$,

e) $p1(X1) = T(2, 0, 0; X2=1)$, and

f) $p2(X1) = T(1, 1, 2; X2=2)$.

Since d = e, either d or e can be eliminated (we eliminate d), the thresholds required for e and f are 1, 1 respectively. Since two adjacent subtrees of X2 are the same (d=e), only 1 threshold is required to fabricate the T-gate in level 1. Therefore, the total number of thresholds for this assignment (X2=first level, and X1= second level) is 3 units. There are only 2! possible orders of variables to be arranged; therefore, the enumerative search has been completed.

11. The best solution is the assignment of X2 = first level and X1 = second level, which requires 2 thresholds. Figure 9 shows two different tree structures that would realize the functional specifications of Example 1.

Example 2.

Consider the 3-valued 3-variable function shown in the map below.

```
       x3=0          x3=1          x3=2
x1 x2 0   1   2     0   1   2     0   1   2
 0 [ 0 | 1 | D ] [ 0 | 1 | 1 ] [ 1 | 2 | 0 ]
 1 [ 0 | 1 | 0 ] [ 2 | 2 | 0 ] [ 1 | 1 | 0 ]
 2 [ 1 | 2 | 0 ] [ 1 | 1 | D ] [ 1 | 0 | D ]
```

The use of Algorithm 1 will produce the realization shown in Figure 10. A complete discussion of this example is given as a part of a User's Guide which is obtainable by writing to the authors.

3.6 Synthesis of Non-Fan-Out Free Networks

In Algorithm 1, the minimization has involved only the identification of trivial residual functions and adjacent residual functions. Several authors [13,14,21] have shown that it is possible to further minimize a network if the T-gates have no fan-out restrictions. Consider Figure 11. If subtrees "a" and "C" are exactly the same, and if there is no fan-out restriction, then either subtree "a" or "c" can be eliminated.

Algorithm 1 includes provisions for the synthesis of non-fan-out-free T-gate networks. Only step 6 as given above needs to be slightly modified, i.e. examine all the subfunctions instead of just adjacent subfunctions. As stated, if adjacent subfunctions are equal, one of the subfunctions can be eliminated and the thresholds and switches of the next level can be reduced. If two non-adjacent sub-functions are equal, one of the
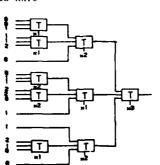
subfunctions can be eliminated. The use of this technique may, or may not, permit a reduction in the next level of the tree.

4.0 Conclusions and Recommendations

The proposed algorithm has been programmed in Pascal for the VAX 11/780. Only minimal attempts were made to optimize the coding of this program. The execution time of this program for various r-valued functions of n variables was tested. Based on this execution time and results projected from Theorem 7, the estimated execution time table of this synthesis program is shown in Figure 12. This execution time is for a fan-out-free network whose input function was completely specified. If an incompletely specified function is being synthesized, the addition of approximately 10% of the average execution time is required. For the non-fan-out-free network, it was found that the average execution time was up to 40% longer than the time for a fan-out-free network.

From the execution time table in Figure 12, one can observe that the execution time grows rapidly once the base or the number of input variables is increased. This is true since the number of comparisons required is $(r^{**}(n-1))n!$, as stated in Theorem 7. This synthesis algorithm is based on an explicit enumeration technique. Yet, many of the multi-valued synthesis procedures behave in the same manner, i.e. they take longer when the base or the number of input variables is increased. At present, there are only a few synthesis methods for multi-valued functions that have been implemented on a computer [15,18,19]. Because these synthesis methods are used to minimize the number of logic gates, not T-gates, no attempt will be made to compare the execution times.

One of the most interesting challenges to this synthesis procedure is the efficient handling of problems with more than six input variables or a base larger than five. As either of these quantities increases, the memory required for an algorithm such as this increases enormously.

Figure 10. T-gate network for example 2.

required 5 thresholds and 10 switches

Or

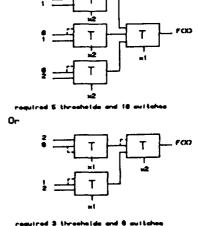required 3 thresholds and 8 switches

Figure 9. Two assignments for Example 1.

Although both of the authors are primarily interested in computer hardware, some future effort will be expended in this software area.

The authors have found incompletely specified functions to be quite common in binary systems. If the same is true in MVL systems, then the inclusion of techniques for optimal choice for don't care conditions in this algorithm will be very valuable.

One of the motivations for developing an algorithm to minimize the component count was the assumption that fewer components would ease the interconnect problem on IC chips. This is an unproven assumption and future work on this project will address the question.

One other incomplete aspect of this research effort is the evaluation of the mode of variable representation that will be most advantageous. The apparent impracticality of IIL circuits for MVL has diverted our attention to voltage mode circuits. However, we do not have any evidence to use in the selection at this time. One of the first circuits that will be attempted is the carry-look-ahead adder for MVL systems.

5.0 References

1. Allen, C. M., and Givone, D. C., A Minimization Technique for Multiple-Valued Logic Systems, IEEE Trans. Comp., Vol. EC-17, 1968, pp. 182-184.
2. Butler, J. A., Fanout-Free Networks of Multi-valued Gates, Proc., 7th ISMVL, May 1977, pp. 39-46.
3. Current, K. W., and Mow, D. A., Four Valued Threshold Logic Full Adder Circuit Implementation, Proc., 8th ISMVL, May 1978, pp. 95-100.
4. Current, K. W., High Density Integrated Computing Circuitry with Multiple-valued Logic, IEEE Trans. Comp., Vol. C-29, February 1980, pp. 191-195.
5. Druzeta, A., Vranesic, Z. G., and Sedra, A. S., Application of Multi-threshold Elements in the Realization of Many-valued Logic Networks, IEEE Trans. Comp., vol. C-23, November 1974, pp. 1194-1198.
6. Higuchi, T., and Kameyama M., Synthesis of Multiple-valued Logic Networks Based on Tree-type Universal Logic Modules, Proc., 1975 ISMVL, May 1975, pp. 121-130.
7. Higuchi, T., and Kameyama, M., Ternary Logic System Based on T-gate, Proc., 1975 ISMVL, May 1975, pp. 290-301.
8. Higuchi, T., and Kameyama, M., Static-hazard-free T-gate for Ternary Memory Element and its Application to Ternary Counters, IEEE Trans. Comp., Vol. C-26, December 1977, pp. 1212-1221.
9. Ishizuka, O., On Multi-valued Multi-threshold Networks Composed of Conventional Threshold Elements, IEEE Trans. Comp., Vol. C-26, No. 12, December 1977, pp. 1251-1257.
10. Ishizuka, O., Synthesis of Multithreshold Tree Networks, IEEE Trans. on Comp., Vol. C-26, No. 12, December 1977, pp. 1294-1297. (Correspondence.
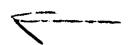11. Kameyama, M., and

Higuchi, T., Synthesis of Multiple-valued Logic Networks Based on Tree-type Universal Logic Module, IEEE Trans. Comp., Vol. C-26, December 1977, pp. 1297-1302. (Correspondence)
12. Kameyama, M., and Higuchi, T., Synthesis of Optimal T-gate Networks in Multiple-valued Logic, Proc., 9th ISMVL, Bath, England, May 1979, pp. 190-195.
13. Kabat, W. C., and Wojcik, A. S., On the Design of 4-valued Digital Systems, Proc., 1980 ISMVL, May 1980, pp. 153-170.
14. McCluskey, E. J., Logic Design of Multivalued IIL Logic Circuits, IEEE Trans. Comp., Vol. C-28, August 1979, pp. 546-599.
15. Moraga, C., Extensions on Multiple-Valued Threshold Logic, Proc., 9th ISMVL, Bath, England, May 1979.
16. Pomper, G., and Armstrong, J. R., Representation of Multivalued Functions Using the Direct Cover Method, IEEE Trans. Comp., Vol. C-30, No. 9, September 1981, pp. 674-679.
17. Singh, A. D., Armstrong, J. R., and Gray, F. G., Combinational and Sequential Multi-valued Logic Design Using Universal Iterative Tree Structures, Proc., 9th ISMVL, Bath, England, May 1979, pp. 182-189.
18. Smith, W. R., Minimization of Multi-valued Functions, Computer Science and Multiple-Valued Logic, Theory and Applications, North-Holland, pp. 221-261.
19. Su, S. and Cheung, T., Computer Simplification of Multivalued Switching Functions, Computer Science and Multiple Valued Logic Theory and Applications, North-Holland, 1977, pp. 189-220.
20. Vranesic, Z. G., Lee, E. S., and Smith, K. C., A Many-valued Algebra for Switching Systems, IEEE Trans. Comp., Vol. C-19, 1970, pp. 964-971.
21. Wojciechowski, W. and Wojcik, A. S., Multiple-Valued Logic Design by Theorem Proving, 9th ISMVL, Bath, England, May 1979, pp. 196-199.
22. Wojcik, A. S., and Fang, K. Y., On the Design of Three-valued Asynchronous Modules, IEEE Trans. Comp., Vol. C-29, October 1980, pp. 889-898.

Figure 11. Non-fan-out-free Network Minimization

Base	No. of Input Variables	Execution Time(sec.)
3	2	0.00017
	3	0.0000
	4	0.000
	5	10.0
	6	100.0
4	2	0.00023
	3	0.000
	4	1.00
	5	31.0
	6	700.5
5	2	0.01020
	3	0.164
	4	3.000
	5	77.10
	6	2014.0

Figure 12. Estimated Execution Time

413

Invited Address

A FUZZY RELATIONAL INFERENCE LANGUAGE FOR EXPERT SYSTEMS

J. F. Baldwin

Engineering Mathematics Department
University of Bristol
England.

Abstract

A fuzzy relational inference language, F.R.I.L.,
is discussed in relation to its application for
designing intelligent knowledge bases and expert
systems. It is a high level query language with
automatic reasoning and is based upon fuzzy set and
relation theories.

Introduction

An expert system is basically a computer soft-
ware system which can emulate a human expert in
storing knowledge about some given subject, make
inferences based on logical deduction, answer queries
and make decisions. It should also show some form
of accountability by being able to present to the
user an argument to justify its choice of answer or
decision. This can take the form of dialogue with
the user. There are many open questions associated
with the design of such systems. How should know-
ledge be represented, what is meant by inference,
how would the various forms of uncertainty (fuzzy,
probabilistic etc) be processed and represented, how
can such systems be made efficient, what computa-
tional model would best exploit future computer
architectures especially parallel processing systems
are only a few of the many questions that we could
ask.

This new computer age with its advanced
V.L.S.I. systems provides cheap computer power -
both processors and memories - , networks, peripherals
etc. Software design must catch up to take full
advantage of these advances. The design of intelli-
gent knowledge bases and expert systems will be one
such advance. It requires the bringing together of
researchers from such fields as artificial intelli-
gence, computer science, systems theory, operations
research, decision theory, control theory, linguis-
tics, etc. Each of these fields has something to
offer to improve both the fundamental design theory
and practical applications of expert systems. Logic
theorem proving, natural language processing,
relational data bases, Baysian decision theory,
search, decision support systems, cluster analysis,
theory of fuzzy sets, learning theory, simulation
and many more general areas of research drawn from
these various fields will all influence the design
of expert systems. Work in this area must be inter-
disciplinary and a new educational programme is
required to give future students a broad education
in all aspects of Information Technology and still
retain a fundamental science and technology educa-
tion.

In this paper we present a new computer language
F.R.I.L. which can be thought of as a high level
language for designing automatic inferential know-
ledge base systems. It stands for Fuzzy Relational
Inference Language and is based upon the mathematics
of relations and incorporates the ability to repre-
sent both fuzzy and probabilistic uncertainties.
The human brain might be thought of as an extremely
complex knowledge base comprising a network of facts
and rules from which new facts can be inferred. In
addition various learning strategies can update this
knowledge with experience. This particular modell-
ing concept is relevant to scientific, technological,
medical, political and economic fields and even to
daily life itself. Decisions are made by analysing
pre-selected facts and rules. This analysis takes
account of various forms of uncertainty and conflic-
ting goals and necessarily includes fuzzy reasoning
when processing the general analytic heirarchy
decision process. The language of F.R.I.L. can be
used to model such processes.

F.R.I.L. has an inherent parallelism which will
allow future computer architectures to be fully
exploited. Its internal automatic inference mech-
anism is not of a search or resolution type associ-
ated with logic inference systems but is more
analagous to Gaussian Elimination for solving
algebraic equations using relational algebra opera-
tions. An implementation of F.R.I.L. is available
on the Honeywell computer at Bristol University. It
is written in MAC-LISP but future implementations in
other languages will be written.

The design of F.R.I.L. includes many ideas in
papers on fuzzy logic and automated inference by
Baldwin (1,2,3,4,5,6), fuzzy systems by Zadeh (11,12,
13,14) with special reference to the paper on PRUF
(15) and Test Score Semantics (16). It has also been
influenced by the work on Codd's Relational Data
Bases - Ullman (9) and Logic Programming - Kawalski
(10).

Some details of F.R.I.L. can be found in Baldwin
and Zhou (7) and Baldwin (8).

The Language of F.R.I.L.

Knowledge representation for the language F.R.I.L. is in the form of Base Relations, Virtual Relations, Set Theoretic Relations and Functions. These are illustrated in the examples which follow. Base relations are tables of facts in which each tuple (row of table) satisfies the relation to some degree X which takes values in the interval (0,1) and represents a fuzzy truth value. Each column is associated with an attribute which can take values from an associated domain. A relation can also be defined as a re-write rule containing other relations and constructs of F.R.I.L. Such a relation is said to be a virtual relation. The logic connectives NOT, AND, OR and their fuzzy logic equivalents can be used in defining virtual relations. A set-theoretic relation is defined by a procedure which takes as input a given tuple of values and returns a truth value in the interval (0,1). For example GREATER(x,y) takes two numerical values x and y and returns 1 if x > y else 0. A function is similar except that it does not have to return a truth value. For example CARD(x) takes a relation name x and returns the sum of the X values of that relation. Set-theoretic relations and functions are written in the host language. The more common ones are provided by the system but the user can define his own.

The system also contains a POINTER construct, computation control constructs such as CONCURRENT, SEQUENCE, REPEAT, CONTROL, an I/O construct MESSAGE and various commands for modifying the knowledge base. These will be illustrated and explained in the examples below.

The user acquires information from the system as follows:

1. The user askes questions through the query language provided by the system. A general query can consist of a sequence of queries and actions.
2. The user input is passed to the multiple command processor which separates multiple queries into single command queries and I/O and is responsible for modifying the knowledge base.
3. Single command queries are passed to the single command processor called the base relation problem generator which translates the query into a problem specification containing base rellations, set-theoretic relations and functions only. This translation involves using the re-write rules of the virtual retlations to re-write virtual relations into base relations. The translation is done in such a way as to provide a problem reduction tree for the next stage of processing. This problem reduction tree breaks down the query into sub-queries if necessary and indicates how the solutions of the sub-queries are to be combined.
4. The problem reduction tree is then passed to the Phase 1 processor which determines a strategy of solution for each sub-query in terms of elementary operations on relations such as 'join', 'projection', 'cross-product', 'difference', 'select', etc. If the relations are fuzzy then the fuzzy equivalents of these elementary operations are used. Phase 1 combines all sub-query solutions into a final solution

which is passed back to the multiple command processor. This processor determines what to do next and if nothing more is to be done, it passes the solution to be printed for the user

Queries take the form of WHICH(...) and DOES(...) queries, a notation borrowed from micro-prolog and contain variables which are bound to attributes. This binding is intuitively obvious and will not be discussed in this paper. Details will be available in the Users Guide to F.R.I.L.

Example 1

Knowledge Base:-

LIKES	NAME	NAME	X
	JIM	IRENE	1
	JOHN	JANE	0.7
	JOHN	MARY	0.6
	HARRY	JILL	0.4
	JILL	TOM	0.2
	IRENE	JIM	0.9
	JANE	JOHN	0.8

TALL	HT	X
I-TYPE	5-9	0
	5-10	0.6
	5-11	0.8
	6-0	1
	7-0	1
	7-0	1

Domain(NAME) = {<character-string>};
Domain(HT) = (4-0,7-0).
N.B. An I-type relation allows for linear interpolation for values between any two values of a given attribute in the relation. The tuple (5-9,|| 0) is included so that interpolated X values in the range 5-9 to 5.10 can be used.

PERSONS	NAME	HT	WT	X
	JIM	6-1	12-0	1
	JOHN	5-9	11-9	1
	IRENE	5-5	10-0	1
	JANE	5-6	9-6	1
	MARY	5-3	8-5	1
	JILL	5-7	9-2	1
	TOM	6-0	13-5	1

HEAVY	WT	X
I-TYPE	10-9	0.1
	11-0	0.2
	11-6	0.4
	11-9	0.7
	12-0	0.8
	13-0	1.0
	14-0	1.0

N.B. In the case of PERSONS, the X column would not normally be included since all X values are 1.
Domain(WT) = (8-0,15-0)
FRIENDS(x,y)←LIKES(x,y) AND LIKES(y,x)
N.B. FRIENDS is a virtual relation and this statement can be interpreted as 'x and y are friends means that x likes y and y likes x'. Variables x,y are local to this definition.

POSS_ATH(x)←PERSONS(x,y,_) AND TALL(y)
HAS_GOOD_FRIENDS(x)←FRIENDS(x,y) AND POSS_ATH(y).

N.B. The interpretation of these statements makes sense as far as the base relations are concerned, even though their realism are open to question. The binding of variables should be obvious from these examples. The underline sign "_" that appears as a relation argument stands for an anonymous attribute variable which does not relate to the definition under consideration.

Consider the query:- Who likes a tall person? In F.R.I.L. this query is written as
WHICH(x LIKES(x,y) AND PERSONS(y,z_) AND TALL(z))
and returns a relation with one attribute NAME.

The problem generator for this example returns

417

the reduction tree

$$REL(x)$$

| |
LIKES(x,y)   PERSONS(y,z,_)   TALL(z)

for which the leaves are base relations.

The Phase 1 process eliminates variables one at a time to produce a succession of reduction trees until the required solution is obtained. For this example $R1(y,z) = Proj_{y \times z} PERSONS(y,z,\_)$ is formed to produce the new reduction tree

$$REL(x)$$

LIKES(x,y)   RL(y,z)   TALL(z)

where R1 is the same relation as PERSONS with the WT column removed.

Next the variable z is eliminated using
1. $R2(y,z) = R(y,z) \sim TALL(z)$ where $\sim$ stands for join.
2. $R3(y) = PROJ_y R2(y,z)$

to produce the new derivation tree

$$REL(x)$$

LIKES(x,y)   R3(y)

where R2 and R3 are given by

R2	NAME	HT	X
	JIM	6-1	1
	TOM	6-0	1

R3	NAME	X
	JIM	1
	TOM	1

The variable y is then eliminated using
1. $R4(x,y) = LIKES(x,y) \sim R3(y)$
2. $R5(x) = PROJ_x R4(x,y)$

to produce the new derivation tree $REL(x) = R5(x)$ where

R4	NAME	NAME	X
	JILL	TOM	0.2
	IRENE	JIM	0.9

R5	NAME	X
	JILL	0.2
	IRENE	0.9

so that R5 gives the solution to the query.

Example 2

Knowledge Base (as for last example).

Query: Name those people with their weights who have a good friend, i.e.

WHICH((x,y) HAS_GOOD_FRIEND(x) AND PERSONS(x,_,y)).

In order to produce the reduction tree, the problem generator scans the query from left to right to produce the first level of the tree which is

| |
HAS_GOOD_FRIEND(x)   PERSONS(x,_,y)

Since PERSONS is a base relation, this node is not expanded. The relation HAS_GOOD_FRIENDS is a virtual relation, so this is further expanded by forming a sub-query and deriving the derivation tree for this sub-query. This is shown in Figure 1. For a good understanding of this method, careful consideration must be given to the concepts of variable renaming and variable binding.

The sub-problems for this query can therefore be represented as

1. To solve for REL(x,y) use
(REL1(x) AND PERSONS(x,_,y))
2. To solve for REL1(x) use (REL2(x,y1) AND REL3(y2))
3. To solve for REL2(x,y1) use
(LIKES(x,y1) AND LIKES(y1,x))
4. To solve for REL3(y2) use
(PERSONS(y1,y2,_) AND TALL(y2))

The Phase 1 procedure is now used in a similar way to the last example to give the solution

REL	NAME	WT	X
	IRENE	10-0	0.9

It should be understood that F.R.I.L. allows 'OR' connectives and more complex connectives can be used and not simply the 'AND' connective used in this paper.

Definition of Pointer

POINTER (<name>,z) sets up a pointer called z which points to the first row of a one attribute relation called <name>. Immediately on reading this command, the problem generator sets up the pointer and finds any z in the query it is processing to this pointer. Any z in the query in the position of a relation name will be changed to the value z is pointing to. Any z in the position of an attribute value of a relation R is changed to =<value> where <value>is that value z is pointing to. The = signifies that only those rows of the relation R for which the corresponding attribute value is <value> are considered. It therefore acts as a selection. The use of = avoids having to have a special notation for variable and constants. After the query is processed, the pointer is moved to the next row and the query repeated. After completion of a query, the position of the pointer is checked to see if there are any more rows. If there are not the query process ends. The answers to each of the queries processed during this self-iteration are cancatinated into the same relation. New headings are introduced if attribute names change during the iteration

Several pointers can be included in the same query or virtual relation definition. These can point to the same or different relations. Only one pointer at a time is moved during the iteration but all possible pointer positions are used.

The X values in rows of <name> are used to modify X values associated with the relation z( ) and R(z) but this modification will not be discussed here, since for all examples below only non-fuzzy relations are spointed at.

A generalisation of the above notation is

POINTER(<name>,(x,y,z)) where <name> is a relation with three attributes. Here (x,y,z) is the pointer with x pointing to first, y to second and z to third column of <name>. During iteration the pointer (x,y,z) is moved as a whole.
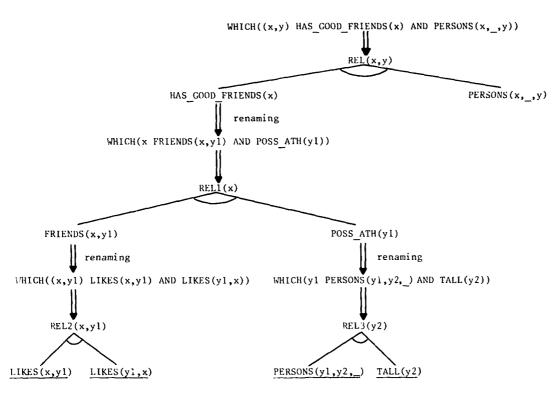
On the other hand
POINTER(<name>,x) and POINTER(x,y) defines a pointer

WHICH((x,y) HAS_GOOD_FRIENDS(x) AND PERSONS(x,_,y))
⟱
REL(x,y)

HAS_GOOD_FRIENDS(x)   PERSONS(x,_,y)
⟱ renaming

WHICH(x FRIENDS(x,y1) AND POSS_ATH(y1))
⟱
REL1(x)

FRIENDS(x,y1)   POSS_ATH(y1)
⟱ renaming   ⟱ renaming

WHICH((x,y1) LIKES(x,y1) AND LIKES(y1,x))   WHICH(y1 PERSONS(y1,y2,_) AND TALL(y2))
⟱   ⟱
REL2(x,y1)   REL3(y2)

LIKES(x,y1)   LIKES(y1,x)   PERSONS(y1,y2,_)   TALL(y2)

Figure 1

x pointing at a row of relation <name> and a pointer y pointing at a relation called by the value pointed to by x.

In the case of
POINTER(<name>,x) AND POINTER(<name>,y)   x and y are both pointers to the same relation <name>. They are moved independently during iteration.

Queries of the form
WHICH((x,y) POINTER(<name>,x) AND x(y))   can be asked. In other words pointer values can be part of the solution.

Example 3

DEPT(NAME) ELEC(COURSES) MATHS(COURSES) COMPUTER (COURSES) µ.PROC(STUDENTS) NETWORKS(STUDENTS) ELECTONICS(STUDENTS) LOGIC(STUDENTS) ANALYSIS (STUDENTS) TOPOLOGY(STUDENTS) INF.TH.(STUDENTS) LANGS(STUDENTS) HARDWARE(STUDENTS)

TRUE	T.V.	X
I-TYPE	0	0
	1	1

VERY	T.V.	X
I-TYPE	0.5	0
	1	1

DEPT_OF_COURSE(x,y)⟵ POINTER(DEPT,x) AND x(y)

STUDENT_ON_COURSE_OF_DEPT(x,y,z)⟵ POINTER(DEPT,z)
                     AND POINTER(z,y) AND y(x)

STUDENT_ON_COURSE(x,y)⟵STUDENT_ON_COURSE_OF_DEPT
                     (x,y,z)

STUDENT_OF_DEPT(x,y)⟵ STUDENT_ON_COURSE_OF_DEPT
                     (x,z,y)

POPULAR_COURSE(x)⟵POINTER(WHICH(w COURSE(w))x) AND

   VERY TRUE(=DIVIDE(CARD(WHICH(y,x) STUDENT_ON_COURSE

   (y,x)),CARD(WHICH(y DEPT_OF_COURSE(z,x) AND

   STUDENT_OF_DEPT(y,z)))))

COURSE(x)⟵ POINTER(DEPT,z) AND z(x)

STUDENT(x)⟵ STUDENT_ON_COURSE_OF_DEPT(x,y,z)

DEPARTMENT(x)⟵ DEPT(x)

NUM_OF_STUDENTS_ON_COURSE(n,z)⟵ POINTER(WHICH(x

   COURSE(x)), z) AND WHERE(n=CARD(z))

COURSE_OF_DEPT(x,z)⟵ POINTER(DEPT,z) AND z(x)

NUM_OF_STUDENTS_IN_DEPT(n,z)⟵POINTER(DEPT,z) AND

   WHERE(n=CARD(WHICH(x STUDENT_OF_DEPT(x,z))))

N.B. CARD(R) returns the cardinality of R, i.e. sum of X values.

Queries:-

WHICH(x COURSE(x)) returns a list of courses that can be studied.
WHICH(x STUDENT(x)) returns a list of students.
WHICH((x,y,z) STUDENT_ON_COURSE_OF_DEPT(y,x,z)) gives

   an inventory of courses with students taking them and associated departments.
WHICH(x DEPARTMENT(x)) returns a list of all departments.
WHICH((y,x) STUDENT_ON_COURSE_OF_DEPT(x,y,=ELEC))

   gives a list of courses and students on them of

419

the ELEC department.

WHICH(y STUDENT_OF_DEPT(=L.BARR,y)) returns the
   department which L.BARR belongs to.

WHICH(y STUDENT_ON_COURSE(=R.YAGER,y)) gives all
   courses which R.YAGER takes.

WHICH(n NUM_OF_STUDENTS_ON_COURSE(n,=NETWORKS))
   gives number of students on Networks course.

For the query - which courses have exactly 2 students
- use

WHICH(x NUM_OF_STUDENTS_ON_COURSE(=2,x))

For the query - do all the Electrical students take
the systems course - use

DOES(POPULAR_COURSE(= SYSTEMS))

The DOES(R) query return the MAX X value of R.

For the query - give popularity of all the courses -
use

DOES(POPULAR_COURSE(x))

Example of Probability Computation

Consider the Markov Chain

STATE	NAME	TRANSITION	FROM	TO	X=PROB
	x1		x1	x1	1/3
	x2		x1	x2	1/3
	x3		x1	x3	1/3
			x2	x1	2/3
			x2	x2	0
PRESENT STATE	NAME X=PROB		x2	x3	1/3
			x3	x1	1/2
	x1 1/2		x3	x2	1/2
	x2 1/2		x3	x3	0

NEW_STATE(x)◄— POINTER(STATE,x) AND DOES

   (MODE (ADD_MULT) AND

   WHICH(z PRESENT_STATE(z) AND TRANSITION(z,x)))

N.B. In this definition MODE(ADD_MULT) changes the
   MAX operator of PROJECTION to an ADDITION
   operator and the MIN operator of JOIN to a
   PRODUCT operator. This allows the X values, in
   this example, to be probabilities rather than
   possibilities.

To obtain 100 iterations of the Markov Chain
state probability distribution one uses

REPEAT 100(MESSAGE "Newstate is" =
   WHICH(x PRESENT_STATE(x));
   UPDATE PRESENT_STATE = WHICH(x NEW_STATE(x)))

In this command, MESSAGE prints to screen the mess-
age given and answer to query. UPDATE changes the
base relation PRESENT_STATE with values from the
query following it.

A Search Example

We illustrate how F.R.I.L. can be used to solve
SEARCH problems - a simple problem is considered but
the same approach can be used in general and modified
to obtain more efficient solution methods.

Three missionaries and three cannibals seek to
cross a river from the left bank to the right bank.
A boat is available which will hold two people and
which can be navigated by any combination of mission-
aries and cannibals involving one or two people. If
the missionaries on either bank of the river are
outnumbered at any time by cannibals, the cannibals
will eat the missionaries. When a boat is moored at
a bank it is counted as part of the bank for these
purposes.

For this example, we use commands

UPDATE <relation name> = WHICH(....)
UPDATE_CHANGES <relation name> = WHICH(...)

which updates the knowledge base. The UPDATE comm-
and has already been discussed. The UPDATE-CHANGES
is similar except only new or changed rows of WHICH
(....) are added to the relation <relation name>.
The details of changed rows depend on keys and will
not be further discussed here. In this example new
rows are added as a result of the WHICH(....) asso-
ciated with the UPDATE-CHANGES. This builds up the
rows of a relation and this is how for this example
the solution is obtained.

Let the state be given by (number of mission-
aries on LEFT Bank, number of cannibals on LEFT Bank,
Bank with Boat))

Such possible states are in relation NODE. A
state can be expanded to produce new possible states
and these are added to NODE. When (0,0,R) is
reached the expansion phase called forward phase
ends and a backward phase then picks out the paths
of the solution. A path is one node connecting
another node in a certain direction. The backward
phase starts in state (0,0,R) as seen by NODEF and
states are added to NODEF by tracing backwards to
the starting state (3,3,L). The paths are recorded
in the relation SOLUTION.

The initial relations PATH and SOLUTION are
empty.

To obtain the solution or decision at any stage,
look up present state in first three columns of
SOLUTION and take corresponding (NM, NC) as the
number of missionaries and cannibals respectively to
put in boat for next journey. The last three columns
of SOLUTION indicates the state resulting from the
next journey.

Forward Phase:-

NODE	M	C	B	BOAT	BANK	BANK	NUM	NM	NC
	3	3	L		L	R		1	0
					R	L		0	1
								1	1
PATH	M	C	B	NM	NM	M	C	B	2	0
									0	2

LEGAL(X1,X2)=((X1,X2)|((x1=0)∨(X1=X2)∨(X1=3))∧
   (0≤X1≤3)∧(0≤X2≤3))

MINIPLUS(x,z,y)——→x+y if z=L; x-y if z=R

NOT_EQUAL(a,b,c,X1,X2,X3)=((a,b,c,X1,X2,X3)|¬
   ((X1=a)∧(X2=b)∧(X3 same as C)))

420

TRANSITION$(M,C,B,x,y,X1,X2,X3)\leftarrow$ NUM(x,y) AND

    LEGAL$(X1,X2)$ AND BOAT$(X3,B)$ AND NODE(a,b,c)

    AND NOT_EQUAL$(a,b,c,X1,X2,X3)$ AND WHERE

    $(X1=$MINIPLUS$(M,X3,x))$ AND WHERE

    $(X2=$MINIPLUS$(C,X3,y))$

FIRST_PHASE$(\ )\leftarrow$ WHICH$((\ )$POINTER$($NODE$,(M\ C\ B))$ AND

    CONTROL(MESSAGE " " = WHICH$((x,y,z)$ NODE(x,y,z)

    AND WHERE$(x=0)$ AND WHERE$(y=0)$ AND WHERE$(z=R))$;

  SEQUENCE(

    UPDATE TEMP=WHICH$((M,C,B,x,y,X1,X2,X3)$

    TRANSITION$(M,C,B,x,y,X1,X2,X3))$;

    UPDATE_CHANGES NODE=WHICH$((X1,X2,X3)$

    TEMP$(M,C,B,x,y,X1,X2,X3))$;

    UPDATE_CHANGES PATH=WHICH$((M,C,B,x,y,X1,X2,X3)$

    TEMP$(M,C,B,x,y,X1,X2,X3)))))$

N.B. UPDATE creates TEMP the first time since it is
not present in knowledge base. CONTROL executes
each command in turn until the first command is com-
pleted for which there is none NULL return from the
WHICH(...) and it then exists. SEQUENCE simply
executes each command of the sequence. The CONTROL
then acts as an IF_THEN_ELSE statement.

Backward Phase:-

NODEF	M	C	B	SOLUTION	M	C	B	NM	NM	M	C	B
	0	0	R									

BACK_PHASE$(\ )\leftarrow$ WHICH$((\ )$ POINTER$($NODEF$,(M,C,B))$ AND

    CONTROL(MESSAGE " " = WHICH$((x,y,z)$NODEF(x,y,z)

    AND WHERE$(x=3)$ AND WHERE$(y=3)$ AND WHERE$(z=L))$;

  SEQUENCE(

    UPDATE TEMP=WHICH$((X1,X2,X3,x,y,M,C,B)$

    PATH$(X1,X2,X3,x,y,M,C,B))$;

    UPDATE_CHANGES NODEF=WHICH$((X1,X2,X3)$

    TEMP$(X1,X2,X3,x,y,M,C,B))$;

    UPDATE_CHANGES SOLUTION=WHICH$((X1,X2,X3,x,y,M,C,B)$

    TEMP$(X1,X2,X3,x,y,M,C,B)))))$

Solution to the M/C problem obtained using

  SEQUENCE(WHICH$((\ )$ FIRST_PHASE$(\ )\ )$;

  WHICH$((\ )$ BACK_PHASE$(\ )\ ))$.

N.B. It must be emplasised that it is the pointer
(M,C,B) that cuases the solution to evolve and the
relations PATH, NODE, solution and NODEF to build up.
Not all solutions will be found since at any stage
of the iteration a node resulting from expansion
which is already in NODE will not be entered or its
corresponding entry in PATH.

Conculsion

The fuzzy relational inference language F.R.I.L.
has been introduced and its use for designing appli-
cations in the general field of knowledge engineer-
ing illustrated. In particular, it can be used for
expert system designs and it has the advantage of
the ability to process fuzzy information.

References

Baldwin, J.F. (1979) A new approach to
approximate reasoning using a Fuzzy
Logic. Fuzzy Sets and Systems, 309-325.
Baldwin, J.F. (1979) Fuzzy Logic and Fuzzy
Reasoning. Int. J. Man-Mach. Stud., 11,
465-480.
Baldwin, J.F. (1979). Fuzzy Logic and its
Applications to Fuzzy Reasoning. In Gupta (Ed.),
Advances in Fuzzy Set Theory and its
Applications, North Holland Pub. Co. 93-115.
Baldwin, J.F. (1981) A Theory of Fuzzy Logic. In
E. Mamdani & B. Gaines (Ed.) Fuzzy Reasoning
and its Applications, Academic Press pp.133-148
Baldwin, J.F. (1981) An Automated Fuzzy Knowledge
Base. In R. Yager (Ed.) Fuzzy Systems,
Pergamon Press.
Baldwin, J.F. & Pilsworth, B.W. (1981) An Infer-
ential Fuzzy Logic Knowledge Base. Proc. Work-
shop on Logic Programming for Intelligent
Systems, R.M.S. Queen Mary, Longbeach, Calif.
Baldwin, J.F. & Zhou, S.Q. (1982) A Fuzzy Relat-
ional Inference Language. (To appear)
Baldwin, J.F. (1983) Proc. I.F.A.C. Fuzzy Infor-
mation Knowledge Representation & Decision
Analysis, Marseille. (To appear).
Ullman, J.D. (1980) Principles of Database Systems,
Pitman.
Kowalski, R. (1979) Logic for Problem Solving,
North Holland Pub. Co.
Zadeh, L. (1965) Fuzzy Sets, Inf. & Control, 8,
338-365.
Zadeh, L. (1973) Outline of a New Approach to the
Analysis of Complex Systems and Decision
Processes, IEEE Trans. Syst., Man & Cybern., 3
28-44.
Zadeh, L. (1975) Calculus of Fuzzy Restrictions.
In Zadeh, Fu & Simura (Ed.) Fuzzy Sets and
their Applications to Cognitive and Decision
Processes. Academic Press.
Zadeh, L. (1978) Fuzzy Sets as a Basis for a Theory
of Possibility, Fuzzy Sets and Systems, 1, 3-28.
Zadeh, L. (1981) PRUF - A Meaning Representation
Language for Natural Languages. In E. Mamdani
& B. Gaines (Ed.), Fuzzy Reasoning and its
Applications, Academic Press.
Zadeh, L. (1981) Test-score Semantics for Natural
Language and Meaning Represnetation via PRUF,
Technical note, 247, S.R.I. International.

Late Paper

PREVIOUS PAGE
IS BLANK

THE SYNTHESIS OF TERNARY FUNCTIONS UNDER FIXED POLARITIES AND TERNARY I²L CIRCUITS

X. Chen and X. Wu

Department of Physics, University of Hangzhou, Hangzhou,
Peoples' Republic of China

Abstract

This paper discusses the expansion of multiple-value functions based upon modulo-algebra and Kronecker product. A transform algorithm of the expansion coefficients of various polarities, and their minimisation are proposed. Ternary function realizations using I²L technology are finally considered.

List of Symbols

n: number of independent variables

$x_i, i = 0$ to $n-1, x_i \in \{0,1,2\}$: independent input variables

$f(x_{n-1},..,x_1,x_0)$, abbreviated to $f(x)$: function of the x_i input variables, $f(x) \in \{0,1,2\}$

$x_i + x_j$: arithmetic sum of x_i and x_j

$x_i \oplus x_j$: mod-3 addition of x_i and x_j

$x_i \cdot x_j$: mod-3 multiplication of x_i and x_j

$\ell_i, i = 0$ to $n-1$: various polarities of variable x_i

$\tilde{x}_i: = x_i \oplus 1$

$\tilde{\tilde{x}}_i: = x_i \oplus 2$

$\overline{x}_i^B$: complement operation on variable x_i,

$$\overline{x}_i^B = \begin{cases} B - x_i \text{ if } B > x_i \\ 0 \text{ otherwise,} \end{cases}$$

$x_i \wedge x_j$: minimum of x_i and x_j

$x_i \vee x_j$: maximum of x_i and x_j

$\overline{x_i \vee x_j}^B$: complement of maximum of x_i and x_j,

$$\overline{x_i \vee x_j}^B = \begin{cases} B - (x_i \vee x_j) \text{ if } B > x_i \vee x_j \\ 0 \text{ otherwise} \end{cases}$$

$U_f(\alpha,\beta,\gamma,x)$: Universal-logic-module for ternary functions, based upon Reed-Muller expansion, i.e. modulo-algebra expansion

$U_h(\xi,\eta,x)$: basic Universal-logic-module for ternary functions, based upon modulo-algebra expansion $U_h(\xi,\eta,x) = \xi \oplus \eta x$

$F]$: column vector whose entries are the 3^n values of $f(x)$, arranged in increasing order of y,
$$y = \sum_{i=0}^{n-1} x_i 3^i$$

$L_B(K)]$: coefficient column vector based upon modulo-algebra expansion under the polarity
$k_{n-1} \ell_{n-1} ... k_1 \ell_1, k_0 \ell_0$, where K, L are the decimal expressions of $k_{n-1}...k_1 k_0$, $\ell_{n-1} ... \ell_1 \ell_0$, respectively

$[T]$: transform matrix from B] to F]

$[T]^{-1}$: transform matrix from F] to B]

$\otimes$: Kronecker product

$$\bigotimes_{i=1}^{n-1} [A_i]: = [A_{n-1}] \otimes .... \otimes [A_1] \otimes [A_0]$$

$[P_k]$: transform matrix of expansion coefficients when $x \rightarrow x \oplus k$

$[{}^\ell I]$: transform matrix of expansion coefficients when $x \rightarrow (\ell + 1)x$

$[P].[T]$: product of matrices over GF(3)

Introduction

Several modulo-algebra expansions of multiple-valued functions have been proposed [1]-[3]. Lately the modulo-algebra expansion of multiple-valued functions over Galois field with q numbers, where q is a prime number or a power of a prime number, has been investigated [4]. We will discuss the transform between F] and the Reed-Muller expansion coefficient column matrix B], and also among various B(K)] under different polarities in section 2 by means of the Kronecker product. Minimal modulo-algebra expansion for ternary functions under fixed polarities is further considered.

According to the modulo-algebra expansion, a Universal-logic-module U_f can be introduced [5], where

$$U_f(\alpha,\beta,\gamma,x) = \alpha \oplus \beta x \oplus \gamma x^2 \tag{1}$$

It has been shown that any ternary function may be realized by the use of only this kind of module. Here in section 3 it will be shown that such a Universal-logic-module $U_f(\alpha,\beta,\gamma,x)$ can be composed of two basic Universal-logic-modules $U_h(\xi,\eta,x)$, where $U_h(\xi,\eta,x) = \xi \oplus \eta x$.

The realization of multiple-valued functions using I²L circuits can be found published [6],[7]. In section 4 a number of further I²L circuits realizing ternary functions are presented.

Modulo-algebra Expansion of Multiple-Valued functions under fixed polarities

It is well known that a single-variable ternary function can be written as:

$$f(x) = b_0 \oplus b_1 x \oplus b_2 x^2$$

Define F] as a column vector of a function f, as previously defined. Define B] as the Reed-Muller coefficient column vector for f based upon modulo-algebra expansion. Define [T] as transform matrix from B] to F].

For a single variable, we have:

$$F] = [f_0 f_1 f_2]^t \qquad (2)$$

where f_0, f_1, f_2 correspond to $f(0), f(1), f(2)$ and

$$f(0) = b_0,$$
$$f(1) = b_0 \oplus b_1 \oplus b_2, \qquad (3)$$
$$f(2) = b_0 \oplus 2b_1 \oplus b_2$$

$$B] = [b_0 b_1 b_2]^t \qquad (4)$$

$$F] = [T].B] = \begin{bmatrix} T_{00} & T_{01} & T_{02} \\ T_{10} & T_{11} & T_{12} \\ T_{20} & T_{21} & T_{22} \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} \qquad (5)$$

Substituting equation (3) into (2), and comparing the result with (5), the transform matrix may be obtained:

$$[T] = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix} \qquad (6)$$

The inverse transform $[T]^{-1}$ is termed transform matrix from F] to B]:

$$B] = [T]^{-1} F] \qquad (7)$$

Similarly, $[T]^{-1}$ can be determined as follows:

$$[T]^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \\ 2 & 2 & 2 \end{bmatrix} \qquad (8)$$

Note, both [T] and $[T]^{-1}$ here are over a single variable. For a two-variable ternary function, we have:

$$f(x_1, x_0) = b_0 \oplus b_1 x_0 \oplus b_2 x_0^2 \oplus b_3 x_1 \oplus b_4 x_1 x_0$$
$$\oplus b_5 x_1 x_0^2 \oplus b_6 x_1^2 \oplus b_7 x_1^2 x_0 \oplus b_8 x_1^2 x_0^2 \qquad (9)$$

In accordance with the Kronecker product and its properties, the above equation may be simplified to:

$$f(x_1, x_0) = [(1 x_1 x_1^2) \otimes (1 x_0 x_0^2)].B], \qquad (10)$$

where $\otimes$ denotes the Kronecker product. It may be proved that there are the following relations:

$$F] = [T]^{\otimes 2}.B] \qquad (11)$$

and

$$B] = \left\{ [T]^{-1} \right\}^{\otimes 2}.F] \qquad (12)$$

where

$$[T]^{\otimes 2} = [T] \otimes [T] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 2 & 1 & 1 & 2 & 1 \\ 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 & 1 \end{bmatrix} \qquad (13)$$

$$\left\{ [T]^{-1} \right\}^{\otimes 2} = [T]^{-1} \otimes [T]^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 2 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad (14)$$

Similarly for n-variable ternary functions, the following equations may be derived:

$$f(x_{n-1}, \ldots, x_1, x_0) = \left[\overset{n-1}{\underset{i=0}{\otimes}} (1 x_i x_i^2) \right].B] \qquad (15)$$

$$F] = [T]^{\otimes n}.B] \qquad (16)$$

$$B] = \left\{ [T]^{-1} \right\}^{\otimes n}.F] \qquad (17)$$

Recalling that the Reed-Muller expansion of a binary function may be derived under different polarities of the input variables, the concept of variable-polarity can be introduced in the multiple-valued function case as well [8,9]. Consider the generalized modulo-algebra expansion of an n-variable ternary function:

$$f(\overset{\bullet}{x}_{n-1}, \ldots, \overset{\bullet}{x}_1, \overset{\bullet}{x}_0) = \left[\overset{n-1}{\underset{i=0}{\otimes}} (1 \overset{\bullet}{x}_i \overset{\bullet}{x}_i^2) \right].B],$$
$$= b_0 \oplus b_1 \overset{\bullet}{x}_0 \oplus b_2 \overset{\bullet}{x}_0^2 \oplus \ldots$$
$$\oplus b_{3^n-1} \overset{\bullet}{x}_{n-1}^2 \ldots \overset{\bullet}{x}_1^2 \overset{\bullet}{x}_0^2, \qquad (18)$$

where $\overset{\bullet}{x}$ is defined as the polarity-expression of x, where it takes a different output value for each possible input value, i.e. there is no loss of information. In ternary system each variable may take six possible polarities, including the variable value itself. They are shown in Table 1.

In the above table the entries in the left column are corresponding modulo-algebra expansions of various polarity-expressions of variable x_i. Note that the last three entries are mod-3 products of the corresponding above entries and constant two. Thus a generalized equation can be obtained:

$$\overset{\bullet}{x}_i = (l \oplus 1).(x_i \oplus k), \qquad (19)$$

x_1	0	1	2
$1 \oplus x_1$	1	2	0
$2 \oplus x_1$	2	0	1
$2x_1$	0	2	1
$2 \oplus 2x_1$	2	1	0
$1 \oplus 2x_1$	1	0	2

Table 1 The polarities of a ternary variable x_1

$$k = 0,1,2 \text{ and } \ell = 0,1$$

Therefore there are $(3!)^n = 6^n$ possible n-variable ternary modulo-algebra expansions. It is said to be the unmodified form when:

$$\ell = 0, \ k = 0$$

The above transform (19) can be divided into two steps. The first transform is termed the k-transform, in which we consider $\ell = 0$. The second transform is termed the ℓ-transform. Now let us consider the k-transform. Take a single-variable function as an example.

Suppose

$$f(x) = b_0^{(k)} \oplus b_1^{(k)}.\overset{\bullet}{x} \oplus b_2^{(k)}.\overset{\bullet}{x}^2$$
$$= [1\overset{\bullet\bullet}{x}x^2].[b_0^{(k)}b_1^{(k)}b_2^{(k)}]^t \tag{20}$$

Consider the k-transform

$$\overset{\bullet}{x} = x \oplus k, \ k = 0,1,2 \tag{21}$$

If $b^{(0)}$ is expressed by b_j, i.e. $b_j \equiv b^{(0)}$, then the standard form ($k = 0$, $\ell = 0$) may be obtained:

$$f(x) = b_0 \oplus b_1 x \oplus b_2 x^2 = \begin{bmatrix} 1xx^2 \end{bmatrix} . \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} \tag{22}$$

From (21) we obtain:

$$(1\overset{\bullet\bullet}{x}x^2) = (1xx^2). \begin{bmatrix} 1 & k & k^2 \\ 0 & 1 & 2k \\ 0 & 0 & 1 \end{bmatrix} \tag{23}$$

and

$$(1xx^2) = (1\overset{\bullet\bullet}{x}x^2). \begin{bmatrix} 1 & 2k & k^2 \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix} \tag{24}$$

If we define

$$[P_k] = \begin{bmatrix} 1 & 2k & k^2 \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix} \tag{25}$$

and

$$[P_k]^{-1} = \begin{bmatrix} 1 & k & k^2 \\ 0 & 1 & 2k \\ 0 & 0 & 1 \end{bmatrix} \tag{26}$$

Substituting (24) into (22) and comparing with (20), we obtain:

$$B^{(K)}] = [P_k].B]$$

Similarly the reverse transform is as follows:

$$B] = [P_k]^{-1}.B^{(K)}]$$

For a n-variable ternary system, 3^n different modulo-algebra expansion coefficient vectors $B^{(K)}]$ can be derived if each variable $\overset{\bullet}{x}_i = x_i \oplus k_i$ takes every possible value, where K is decimal number of ternary $k_{n-1}...k_1 k_0$, $K = 0,1,...3^n-1$.

The following equation may be obtained by using a Kronecker product and the equation (25):

$$B(K)] = \left\{ \overset{n-1}{\underset{i=0}{\otimes}} [P_{k_i}] \right\}.B] \tag{27}$$

It is obvious that the complexity of the various modulo-algebra expansion coefficient vectors $B^{(K)}]$ of a function varies with K. The more zero-coefficients in $B^{(K)}]$, i.e. the fewer the necessary product-terms in the modulo-algebra expansion, the simpler the function form.

Example 1

Consider a two-variable ternary function expressed by Fig.1. From the K-map its column vector is

$$F] = 0\ 2\ 0\ 2\ 1\ 0\ 0\ 2\ 0]^t$$

Fig.1 Example 2-variable ternary function

The unmodified expansion coefficient vector can be obtained from the equations (12) and (14):

$$B] = 0\ 1\ 1\ 1\ 2\ 1\ 1\ 2\ 1]^t$$

Therefore its modulo-algebra expansion is as follows:

$$f(x_1,x_0) = x_0 \oplus x_0^2 \oplus x_1 \oplus 2x_1 x_0 \oplus x_1 x_0^2$$
$$\oplus x_1^2 \oplus 2x_1^2 x_0 \oplus x_1^2 x_0^2$$

When the eight other possible polarities are considered, it is found that there are the least non-zero coefficients when $k_1 = 2$, $k_0 = 1$, i.e. $K = 7$.

The corresponding expansion coefficient vector can be derived from the equations (25) and (27):

$$B^{(7)}] = ([P_2] \otimes [P_1]).B] = \begin{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{bmatrix}.B$$

$$= \begin{bmatrix} 1\,2\,1 & 1\,2\,1 & 1\,2\,1 \\ 0\,1\,1 & 0\,1\,1 & 0\,1\,1 \\ 0\,0\,1 & 0\,0\,1 & 0\,0\,1 \\ 0\,0\,0 & 1\,2\,1 & 2\,1\,2 \\ 0\,0\,0 & 0\,1\,1 & 0\,2\,2 \\ 0\,0\,0 & 0\,0\,1 & 0\,0\,2 \\ 0\,0\,0 & 0\,0\,0 & 1\,2\,1 \\ 0\,0\,0 & 0\,0\,0 & 0\,1\,1 \\ 0\,0\,0 & 0\,0\,0 & 0\,0\,1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Its corresponding modulo-algebra expansion therefore is as follows:

$$f(x_1,x_0) = 2\widetilde{x}_0 \oplus \widetilde{x}_1^2.\widetilde{x}_0^2 .$$

After choosing the optimum K, further consider the determination of optimum L, where L is the decimal expression of binary $\ell_{n-1}..\ell_0$, $\ell_i \in (0,1)$. Similarly, we may obtain transforms corresponding to a replacement of $\hat{x} = (\ell \oplus 1).x$. For a single-variable ternary function, we find:

$$\begin{aligned} B^{(K)}] &= [{}^\ell I].{}^\ell_B{}^{(K)}] \\ {}^\ell_B{}^{(K)}] &= [{}^\ell I]^{-1}.B^{(K)}] \end{aligned} \right\} \quad (28)$$

where

$$[{}^\ell I] = [{}^\ell I]^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+\ell & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (29)$$

It can be seen that the ℓ-transform does not change the number of non-zero coefficients, since all diagonal elements are non-zero and the others are zero in $[{}^\ell I]$. For n-variable ternary function, we derive

$$L_B{}^{(K)}] = \left\{ \overset{n-1}{\underset{i=0}{\otimes}} \left[\ell_i I\right] \right\}.B^{(K)}]$$

$$B^{(K)}] = \left\{ \overset{n-1}{\underset{i=0}{\otimes}} \left[\ell_i I\right] \right\}.L_B{}^{(K)}] \quad (30)$$

Although the number of non-zero coefficients remains unchanged, the number of coefficients with value 2 varies with different L. The fewer the number of these coefficients, the simpler is the corresponding modulo-algebra expansion of a function. Therefore the optimum procedure can be stated as follows. Firstly search for the optimal K value under the K-transform so that the number of non-zero coefficients is minimised. Then determine the optimal L value under L-transform so that the number of coefficients being two is minimum. It may be seen that only $3^n + 2^n$ searches are necessary for any n-variable ternary function, against 6^n exhaustive searches. Consider the above Example_1, we can find the number of coefficients is being two is minimum when L = 1, i.e. $\ell_1 = 0$, $\ell_0 = 1$. The corresponding coefficient vector is:

$$L_B(K)] = {}^1_B(7)] = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^t$$

The corresponding modulo-algebra expansion is:

$$f(x_1,x_0) = (2\widetilde{x}_0) \oplus (\widetilde{\widetilde{x}}_1)^2.(2\widetilde{x}_0)^2$$

Note that in this case the non-zero coefficients of all products are of unity value.

Universal-Logic-Modules

Hurst and Tokmen have disclosed a Universal-logic-module based on modulo-algebra for a ternary system [5]:

$$U_f(\alpha,\beta,\gamma,x) = \alpha \oplus \beta x \oplus \gamma x^2 \quad (31)$$

This may be decomposed into smaller cells:

$$U_f(\alpha,\beta,\gamma,x) = \alpha \oplus \beta x \oplus \gamma x^2 = \alpha \oplus x(\beta \oplus \gamma x) = \alpha \oplus U_h x$$

Here $U_h(\xi,\eta,x) = \xi \oplus \eta x$ (32)

It is obvious that a $U_f(\alpha,\beta,\gamma,x)$ can be realized by two U_h cells as shown in Fig.2. Therefore U_h forms a complete set. The advantages using U_h are a simpler algebraic expression and flexibility in employment. After examining all twenty-seven single variable ternary functions, it can be shown that in addition to constants 0, 1 and 2 and variable x itself, nine of them may be realized by only one U_h. They are:

$f_1(x) = x^2$, $(0,x)$ $\quad$ $f_4(x) = x \oplus x^2$, (x,x)

$f_6(x) = 2x$, $(0,2)$ $\quad$ $f_{10}(x) = 1 \oplus x^2$, $(1,x)$

$f_{12}(x) = 1 \oplus x$, $(1,1)$ $\quad$ $f_{15}(x) = 1 \oplus 2x$, $(1,2)$

$f_{19}(x) = 2 \oplus x^2$, $(2,x)$ $\quad$ $f_{21}(x) = 2 \oplus x$, $(2,1)$

$f_{24}(x) = 2 \oplus 2x$, $(2,2)$

Fig.2 Implementation of U_f using two U_h modules

The values in brackets express the corresponding input patterns. Note five of nine are merely polarities variation of variable x. They are

$$f_{12}(x) = 1 \oplus x, = \tilde{x}, \quad f_{21}(x) = 2 \oplus x, = \tilde{\tilde{x}}, \quad f_{6}(x) = 2x,$$

$$f_{24}(x) = 2 \oplus 2x, = 2\tilde{x}, \quad f_{15}(x) = 1 \oplus 2x, = 2\tilde{\tilde{x}}.$$

Thus the various polarities of variable x can be realized using a single U_h.

Realization of ternary logic using I^2L circuits

The basic operations in modulo-algebra are mod-3 addition, and mod-3 multiplication [10]. They may be reduced to addition, multiplication and mod-3 limit. Here we consider the realization of some ternary logic circuits.

Polarity-transform circuits

Figs.3(a)-(f) give various polarity-transform circuits. Fig.3(a) is a complement circuit; Fig.3(b) is a circuit for multiplication by two. Fig.3(c)-(f) show how other polarity-transform circuits may be obtained in terms of appropriate serial connection of the above two circuits. The function of Figs. 3(e) and (f) is the same, but the circuit of (e) is simpler.

Mod-3 addition

Mod-3 adder is shown in Fig.4. It can be seen that the upper branch creates arithmetic addition, the lower branch being a mod-3 limiter.

Mod-3 multiplication

The difficulty of realizing multiplication of two variables consists in the physical interpretation of multiplicand and multiplier. If product and
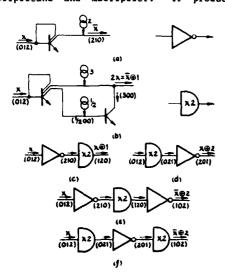
Fig.3 I^2L polarity-transform circuits
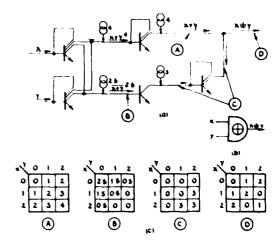
Fig.4 Mod-3 adder realisation
(a) circuit, (b) legend, (c) K-maps

multiplicand are expressed by using the same physical measure, say electrical current, then the multiplier becomes a pure number without any physical meaning. However, up to now any simple effective control of amplification in terms of current has not been found. In a binary system, this difficulty is avoided because the operation of multiplication and the minimum of two variables 0,1 is the same.

We extend this interpretation into the ternary system. Fig. 5(a) illustrates that the multiplication of two variables can be divided into two operations. The central K-map of Fig.5(a) denotes minimum of variables, i.e. $x \wedge y$. Since $x \wedge y = \overline{\bar{x} \vee \bar{y}}$, this may be implemented as shown in Fig.5(b). The RH K-map expresses $\bar{2}^{(x+y)}$, i.e.

$$\bar{2}(x+y) = \begin{cases} x + y - 2, & \text{if } x + y \geqslant 2 \\ 0 & \text{otherwise} \end{cases}$$

Fig.5(b) gives the total realization. The upper part implements $x \wedge y$; the centre realizes $\bar{2}(x+y)$, and the lower is the mod-3 limiter. The functions of various points A – D are expressed in the corresponding K-maps shown in Fig.5(d).

Arithmetic circuits

On the bases of mod-3 addition and mod-3 multiplication, full-adder and full-multiplier with carry may be designed. Figs.6(a) and (b) show their I^2L circuits respectively, where C is carry input and C' is carry output.

U_h circuit

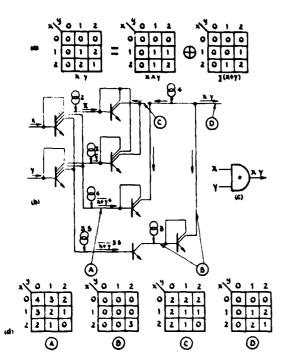This may be realized by the cascade of a mod-3 multiplier and a mod-3 adder, as shown in Fig.7.

428

Fig.5 Mod-3 multiplier realisation

Fig.6 Full-adder and full-multiplier realisations
(a) adder, (b) multiplier

Fig.7 Basic universal-logic-module U_h

Conclusions

The modulo-algebra expansion of a ternary function has been discussed in terms of modulo-algebra and Kronecker product. Because of the six possible polarities of a ternary variable the minimization of ternary functions is more complicated than that of the corresponding binary case. A minimization procedure has been suggested. Since the mod-3 multiplier costs are high, the first step is a search for the optimum K to make the number of non-zero expansion coefficients minimum. On the basis of this new function, we then find the optimum L corresponding to the minimum number of product terms which have coefficients of two. Such a searching process can be implemented by using a computer search.

A basic Universal-logic-module, two of which are capable of realizing any single-variable ternary function, has also been considered. A number of I^2L circuits realizing various ternary functions have been proposed.

References

1. Berlin, R.D., "Synthesis of N-valued switching circuits", IRE Trans., EC-7, 1958, pp.52-56.

2. Lowenschuss, O., "Non-binary switching theory", IRe Natn. Conv. Rec. 6(4), 1958, pp.305-317.

3. Tamari, D., "Some mutual applications of logic and mathematics", Proc. 2nd Int. Colloq. of Mathematical Logic, 1952, pp.89-90.

4. Green, D.H. and Taylor, I.S., "Modular representation of multiple-valued logic systems", Proc.IEE, 121, 1974, pp.409-418.

5. Tokmen, V.H. and Hurst, S.L., "A consideration of Universal-logic-module for ternary synthesis, based upon Reed-Muller coefficients", Proc. 9th Int. Symp. MVL, 1979, pp.248-256.

6. Dao, T.T., McCluskey, E.J. and Russell, L.K., "Multi-valued integrated injection logic", IEEE Trans., C-26, 1977, pp.1233-1241.

7. McCluskey, E.J., "Logic design of multi-valued I^2L logic circuits", IEEE Trans., C-28, 1979, pp.546- 559.

8. Wu, X., Chen, X. and Hurst, S.L., "Mapping of Reed-Muller coefficients and the minimisation of exclusive-OR switching functions", Proc.IEE Part E, 129, 1982, pp.15-20.

9. Green, D.H. and Taylor, I.S., "Multiple-valued switching circuit design by means of generalised Reed-Muller expansions", Digital Processes,2, 1976, pp.63-81.

10. Hurst, S.L., "Logical Processing of Digital Signals", Crane Russak, N.Y. and Edward Arnold, London, 1978.

Author Index

END

FILMED

4-85

DTIC